POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Artificial Intelligence for Security Attacks Detection

Relatori
Prof. Antonio Lioy
Dr. Ing. Diana Berbecaru
Dr. Ing. Daniele Canavese

Stefano Giannuzzi

Dicembre 2022

Acknowledgements

I want to express my gratitude to Prof. Lioy, Dr. Ing. Berbecaru and Dr. Ing. Canavese for their assistance and guidance with this thesis. Without their encouragement and direction, this work would not have been possible. When I had problems, they were always there to help me.

I want to thank all my family and close friends for their constant support throughout my master's program.

Contents

Chapter 1

Introduction

Nowadays, cyber-attacks are becoming more frequent and use new threats that are unknown to the common defence system. These techniques can be so sophisticated that they go unnoticed by Intrusion Detection Systems (IDS), which detect malicious activity on a system or a network. As reported by Kaspersky, there is an increase of half in the cybersecurity incidents[1]. This also depends because many organizations do not have appropriate defence systems or those systems do not recognize unknown threats without additional resources. Moreover, according to a study conducted by Positive Technologies among financial, fuel and energy organizations, government agencies, industrial enterprises, and other sectors, an external attacker can breach a network perimeter and access local network resources in 93% of cases[2]. For this reason, there is a need to use new systems that find the "anomaly" in the network.

There are two main techniques for IDS: signature-based and anomaly-based. The IDS signature-based system typically only detects attacks already discovered for which a signature exists and is stored in a database. As a result, any traffic that matches any signature in the database containing all known attack signatures is categorized as belonging to the corresponding attack. IDS Anomaly-based methods can identify attacks that have not yet been discovered without registered signatures using Machine Learning and Deep Learning techniques. The concept of Anomaly-based is more similar to whitelisting; when the system detects behaviour outside an acceptable range, that is an anomaly.

The Anomaly-based system is pretty different from the IDS Signature-based because the last just compare signatures with events that have already happened, and only the traffic that generates a match is reported. Anomaly-based detection searches for anomalous traffic, out of the ordinary, in which there are also new threats that have not yet been discovered and for which there is not yet a signature. For this reason, an anomaly-based IDS system has advantages compared to an IDS signature-based system, especially on new attacks.

---

[1] https://www.kaspersky.com/about/press-releases/2022_share-of-high-severity-cybersecurity-incidents-facing-organizations-increases-by-half-in-a-year

[2] https://www.ptsecurity.com/ww-en/about/news/positive-technologies-cybercriminals-can-penetrate-93-of-local-company-networks-and-trigger-71-of-events-deemed-unacceptable-for-their-businesses/

1.1 Objective

The main issue is that many anomaly-based IDS models are insufficiently exhaustive and do not recognize unknown attacks. This occurs because these models are mainly supervised models that need to classify traffic based on what they learned during the training phase, but if there is an unknown attack, their accuracy decreases. This happens because they are not trained to do so.
Therefore, there is a need to create a new model that can detect unknown attacks in network traffic.

The main objective of this thesis is to identify anomalies in network traffic using artificial intelligence techniques. More specifically, a model based on an Auto-Encoder is proposed in this thesis to recognize unknown attacks based on flow characteristics.

Compared to other works utilizing the same strategy, the model proposed in this dissertation detects the same attacks with the same dataset more effectively.
During the testing phase, the proposed model is compared to supervised learning models such as Random Forest and Extreme Gradient Boosting to assess its ability to detect known and unknown attacks.

The first test is conducted on the CIC-IDS2017 dataset to evaluate the model's ability to detect known attacks. That means the same attacks are present in both the training and testing dataset. In this phase, the supervised model outperforms the proposed model because they are trained to recognize the attacks. However, for some attacks, the proposed model achieved the same results as the supervised models.
In the second test, the model's ability to detect unknown attacks is evaluated with the TORSEC dataset, meaning that they are trained without considering a specific attack and then tested on it. For instance, models are trained on all DoS attacks (DoS Slowhttptest, DoS hulk, and DoS GoldenEye) with the label "DoS", and then tested on a variant of DoS attacks called DoS Slowloris. In this case, the proposed model outperforms the supervised models in detecting the unknown DoS attack Slowloris, which behaves as random.

In addition, the proposed model is evaluated on a new dataset, the Hearthbleed dataset described in Chapter 5. This is done because, in the initial test, the auto-encoder achieved high values for detecting the Heartbleed attack; consequently, this dataset is created ad hoc to comprehend the motivations.

1.2 Outline

The thesis is structured as follows:

- Chapter 2 describes the related studies.
- Chapter 3 is divided into two sections, the first introduces the basic concept of network traffic (IP, TCP, TLS), the most common network attacks, and the systems used to detect and mitigate them (IDS, IPS); then the second introduces the basic concepts of Machine Learning and Deep Learning.
- Chapter 4 describes motivations and the contribution of this thesis.
- Chapter 5 analysed the datasets used for this thesis.
- Chapter 6 describes the design of the proposed solutions and the features of the dataset used.

- Chapter 7 describes the implementation of the proposed model.
- Chapter 8 presents the results obtained from the proposed model and then compared to the supervised models (Random Forest and Extreme Gradient Boosting) and the model described in the reference paper.
- Chapter 9 summarizes the workflow of this thesis, and the result obtained. In the end, there are the final observations and future works.
- Appendix A shows the features of the CIC-IDS2017.
- Appendix B shows the histograms used for feature selection.
- Appendix C is the user manual to use the provided solution.
- Appendix D is the developer manual to modify the provided solution.

Chapter 2

Related Works

This section explains the state of the art of the different works done by researchers to complete this thesis.

2.1 Intrusion detection system for atypical cyberattacks

This work, as described [1], proposes a defensive AI engine that combines a Heterogeneous Feature Selection Ensemble (HFSE) technique, an ensemble model used to select features, with AI models with hyperparameter optimization.

In this study, they used a Heterogeneous Feature Selection Ensemble (HFSE) composed of seven different models that select features with a voting operation, then they developed the proposed approach, a Stacked Decision Tree Classifier (S-DTC) for identifying binary attack flows. A stacked decision tree is an ensemble model composed by different decision trees[1] one after the other. The AI models are trained and validated using the CIC-IDS2017 data set.

The system is subsequently tested by simulating real-world scenarios with generated attack flows.
Using different Deep Learning and Machine Learning models, they demonstrate the effectiveness of the suggested unusual attack flow detection method (S-DTC).
The suggested defensive AI engine significantly improves the True Positive Rate ($TPR = \frac{TP}{TP+FN}$)[2] of AI models against many atypical attacks, as demonstrated by simulation results.
Using the proposed Defensive AI Engine, they compare and evaluate several AI models such as DNN, L-SVC, and S-DTC against atypical attacks through a comprehensive experimental analysis.

These AI models are trained using a training set and tested using generated attacks with different feature profiles. Experiments show that after hyperparameter[3] optimization (HPO), the performance of AI models trained with their methodology is considerably increased against both normal and atypical attacks.

---

[1] tree model used for classification and regression.

[2] the probability that an actual positive test positive.

[3] parameter which controls the model learning process.

They propose an innovative defensive AI engine to build and test IDS models against dynamically changing, atypical attacks in order to improve current knowledge of such attacks.

Multiple performance metrics are used to evaluate the AI model sensitivity to detect atypical attacks. These metrics showed that the S-DTC achieved higher values than the other models, in fact it obtained a value of 1.0 for accuracy ($accuracy = \frac{TP+TN}{TP+TN+FP+FN}$), precision ($\frac{TP}{TP+FP}$), recall ($\frac{TP}{TP+FN}$) and F1-score ($\frac{2 \times precision \times recall}{precision+recall}$).
They observe an increase in the performance of their proposed AI models against different atypical attacks, they require more hyperparameter optimization and training to classify True Positive Rate (TPR) accurately.

The primary focus of this research is on attacks that can change into atypical attacks, i.e., beginning as a known attack and then transforming into different profiles that can fool the IDS. They utilized Slowloris and SlowHttptest, two kinds of slow-rate Denial-of-service assaults, to generate atypical attacks for IDS evaluation.

The aspect that must be emphasized in this work is that they have tested the system on a model trained with a variant of the attack.

2.2 Anomaly-based intrusion detection using variational auto-encoder and autoencoder

This work is described in the paper [2].

Using a semi-supervised learning strategy, the performance of Auto-Encoder (AE) and Variational Auto-Encoder (VAE)[4] algorithms along with One-Class Support Vector Machine (OCSVM)[5] are evaluated in this study.
Only "benign" data flow are utilized in the construction of the models, from the dataset CIC-IDS2017[6] [3]. In addition, the models are evaluated using both normal and anomaly data.

For AE and VAE, the encoder and decoder have two hidden layers with 512 and 256 dimensions each respectively. Both AE and VAE have 64 dimensions in their bottleneck layer. In order to determine the best-performing models of both AE and VAE, the hyperparameters are determined by trial and error while maintaining a constant number of layers.

Initially, the features of the dataset are normalized using the feature scaling approach, also known as unity-based normalization, to scale all values in the range [0,1].
In the testing phase, all the labels other than "benign" are called "anomaly", and then the metrics are computed using only the "normal" and "anomaly" records of the selected attack class (DoS Slowloris, FTP Patator, Dos Goldeneye, ...), without considering the records of any other attack classes.

In order to determine whether the approaches can distinguish unknown attacks from normal traffic, a second evaluation strategy combines all attacks into the class "anomaly", and then it is distinct from "normal" using the above strategies (AE, VAE, OCSVM).

[4] Deep Learning model which learns to compress and reconstruct data.

[5] Machine Learning model which has the object to find a hyperplane that classifies the data.

[6] https://www.unb.ca/cic/datasets/ids-2017.html

The experimental outcomes are calculated using ROC-curve (Receiver Operating Characteristic) and AUC (Area Under the Curve) measures. Based on the results, the detection rate of VAE (Variational Autoencoder) is often superior to that of AE and OCSVM. Observe that the AUC for all attacks is 75.96% for the VAE and 73.38% for the AE. Regarding all the attacks, the distinctions between VAE and AE are negligible, though in the advantage of VAE.

However, it is also essential to note that the methods must be complemented by supervised learning techniques due to their high rate of false positives. In addition, in order to boost the detection rate of methods, the flow-based characteristics acquired at certain time intervals can be considered, as the characteristics of certain attacks can be better modelled.

2.3 CSE-IDS: deep learning and ensemble models for network-based intrusion detection systems

The proposed CSE-IDS, described in [4], consists of three layers and accurately classifies network traffic using Deep Learning algorithms and Ensemble techniques. The first layer is a Cost-Sensitive Deep Neural Network (CS-DNN), the second is an Extreme Gradient Boosting (XGBoost), and the third is a Random Forest (RF).

Seven additional NIDSs[7] (DNN, XGBoost, RF, Siam-IDS [5], I-SiamIDS [6], LIO-IDS [7]) are compared to CSE-IDS using a variety of evaluation measures, including Accuracy, Recall, Precision, F1-score, ROC curve, AUC values, and computational duration.
The collected results demonstrate that the proposed CSE-IDS has the best performance in intrusion detection and effectively manages class imbalance in NIDS.

2.4 Meta-Learning to improve unsupervised intrusion detection systems

This paper [8] methodically implements several meta-learning approaches using ensembles of unsupervised base-learners and discusses how the adoption of a particular meta-learning approach can considerably reduce the frequency of miss-classifications in comparison to non-meta unsupervised algorithms.

After elaborating on different meta-learners and their effectiveness for detecting (zero-day) attacks in Cyber-physical systems (CPS), they conduct an experiment to compare various techniques.

The training set is collected from publicly available datasets that report on network intrusion detectors and biometric authentication systems, which are typically used to improve CPS security.
The unsupervised algorithms chosen for this research will be utilized both as non-meta learners and as base classifiers of meta-learning techniques to detect unknown threats or zero-day attacks.
They choose a total of 21 datasets, 15 unsupervised algorithms, and 9 distinct meta-learning techniques, which we instantiate by taking the aforementioned unsupervised

[7]Network Intrusion Detection System: identify cyber attacks on a computer network.

algorithms as base-learners.
In 20 of the 21 datasets utilized for this research, they show that meta-learning decreases miss-classifications, consequently improving metric scores.

They first divide the supervised and unsupervised learning models according to their performances:

1. G1 Higher Matthews Coefficient (MCC)[8] (SVM (Classification family), ODIN [9] (Neighbor-based), FastABOD [10] (Angle/Neighbor-based).

2. G2 Higher Accuracy (SDO [11] (Density family), DBSCAN [12] (Clustering), SVM (Classification)).

3. G3 Higher Recall (SDO [11] (Density family), DBSCAN [12] (Clustering), SVM (Classification)).

4. G4 Seven Algorithms, one for each family (SVM (Classification family), ODIN [9] (Neighbor-based), FastABOD [10] (Angle/Neighbor-based), SDO [11] (Density), DBSCAN [12] (Clustering), SOM (Neural Networks), HBOS (Statistical))

Then, they build three different meta-learning techniques that were used for each of the above groups:

- Single Classifier (SC) Meta-Learners. More specifically, for each algorithm, they instantiate Bagging meta-learners.

- Multiple Classifiers (MC) Meta-Learners. (Weighted) Voting and Stacking (Generalization) based on the above algorithms.

- Multiple Classifiers with Ordering (MCO) Meta-Learners. Delegating, Cascading, and Cascade Generalization for each algorithms group.

In the end, they performed the experiments with Multiple Classifiers (MC) and Multiple Classifiers with Ordering (MCO) meta-learners using G1 to G4 groups of base learners.

They discovered that selecting classifiers from distinct families, described in MCO meta-learners by conducting experiments with different base groups, does not guarantee that has the best performance. Moreover, groups of base learners with similar characteristics performed poorly. Not all groups with "diverse" base-learner algorithms performed well.

The MCC score obtained by meta-learners is higher than the MCC of unsupervised algorithms for 20/21 datasets, indicating that meta-learners guarantee less misclassification in nearly all circumstances and none for one dataset.
In 11 on 21 datasets, boosting beats unsupervised algorithms and other meta-learners, hence lowering miss-classifications.

[8] Also know as Phi Phi Coefficient, it is the association between two binary variables.

2.5 Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study

In this research, described in [13], they compared different Deep Learning and Machine Learning approaches for Intrusion Detection Systems. In particular, they compared seven Deep Learning models and Machine Learning models.

These models approaches are compared using two new datasets, the CSE-CIC-IDS2018[9] and the Bot-IoT[10] datasets, with three crucial performance indicators: False Alarm Rate (FAR)[11], Accuracy (ACC), and Detection Rate (DR)[12].

They demonstrate that the Convolution Neural Networks (CNN) have the highest True Negative Rate (TNR)[13] and the highest Detection Rate for Distributed Denial of Service (DDoS) attacks and Botnet. The Recurrent Neural Networks (RNN) have the highest Detection Rate for Web Brute Force attacks, Denial of Service (DoS) attacks and Infiltration. About unsupervised models, DBN has the highest True Negative Rate (TNR) and the highest Detection Rate for Brute Force Web attacks, DoS Hulk attacks and DDoS attacks. The Auto-Encoders (AE) has the highest Detection Rate for Web Brute Force attack, DoS Slowloris, and Infiltration. The Deep Boltzmann Machines (DBM) give the highest Detection Rate for DoS Hulk, DoS SlowHTTPTest, DoS GoldenEye, DDoS and Botnet.

About Accuracy the CNN has the higher accuracy for CSE-CIC-IDS2018 and the Bot-IoT. For the unsupervised, the higher accuracy is achieved by AE for CSE-CIC-IDS2018 and for Bot-IoT.

Compared to the Machine learning Model, the CNN model has the highest overall detection rate (DR Overall) in the CSE-CIC-IDS2018 dataset and in the Bot-IoT dataset. In the unsupervised models, the AE model has the highest overall detection rate (DR Overall) in CSE-CIC- IDS2018 dataset and in the Bot-IoT dataset.

2.6 Using machine learning techniques for DoS/DDoS attacks detection

This research [14] describes the Smart Detection system, an online solution to DoS/DDoS attack detection that uses the Random Forest Tree algorithm to classify network traffic based on samples taken directly from network devices using the sFlow protocol [14]. Several tests were performed to calibrate and evaluate the performance of the system.
The suggested system was assessed using three intrusion detection benchmark datasets, namely CIC-DoS, CIC-IDS2017, and CSE-CIC-IDS2018.

[9] https://www.unb.ca/cic/datasets/ids-2018.html

[10] https://research.unsw.edu.au/projects/bot-iot-dataset

[11] Also known as false positive rate, it is the ratio between the number of negative events considered as false positives and the total number of negative events $\frac{FP}{FP+TN}$

[12] It is also known as True Positive Rate.

[13] Also called Specificity, it is the probability of a negative test to be really negative

[14] https://sflow.org/sflow_version_5.txt

They evaluate the method with Detection Rate (DR), False Alarm Rate (FAR), Precision (PRC) and F1-Score (F1). The model proposed in the paper obtain higher performance for CSE-CIC-IDS2018 than for CIC-IDS2017 and CIC-DoS.

Chapter 3

Background

3.1 Basic Concepts of Networks

This section describes the basic concepts of Networks.

In the first part, there is a description of the Network Layer and the Transport Layer. Then, the Cyber attacks that will be used by the proposed model are described. In the end, there is a description of what an Intrusion Detection System (IDS) is, what are the types of IDS and the difference between Signature-based and Anomaly-based with their advantages and disadvantages.

3.1.1 Internet Protocol (IP)

As described in [15], in this subsection there is a focus on key aspects of the Network Layer, mainly on the Internet Protocol (IP)[16].

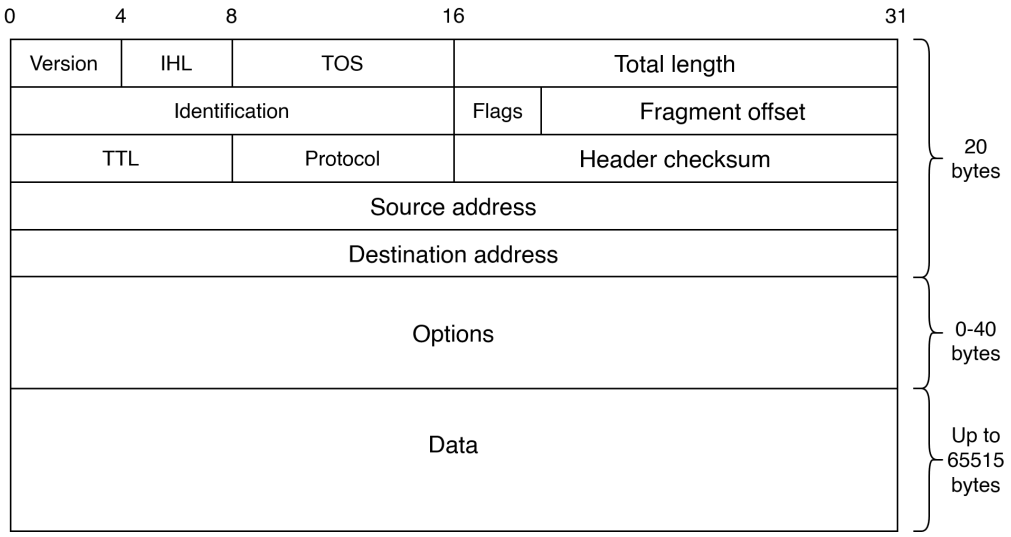


Figure 3.1: It represents the IPv4 header. At the top is represented the number of bits that each field occupies, on the side, it indicates the length in bytes of the selected fields.

The Figure 3.1 describes the IPv4 header:

- Version [4 bit]: identifies the version of IP (IPv4, IPv6).

- Internet Header Length (IHL) [4 bit]: identifies the length of the IP packet header, it is required for the "Options" field which makes a variable length header.
- Type of Service (TOS) [8 bit]: used to differentiate traffic on a device, for example, if you want to prioritise some particular data.
- Total Length [16 bit]: identifies the total length of the package in bytes.
- Identification [16 bit], Flags [3 bit] and Fragment Offset [13 bit]: used for the fragmentation of the packet. These fields are used to rebuild a fragmented package and to prevent the loss of a fragment.
  Today practically all local networks are made with Ethernet where the maximum transfer unit (MTU) is 1500 bytes [17].
- Time to live (TTL) [8 bit]: Maximum number of routers that the packet can fit before its destruction. When this happens an ICMP packet with a "time exceeded" message is sent to the source.
- Protocol [8 bit]: indicates the level 4 protocol contained in the payload (i.e. TCP).
- Header Checksum [16 bit]: error check on bits. The checksum is done only on the header part. Each router calculates the checksum on the header and compares it with that in the packet, if they are different there was some error in the transmission and the packet is discarded.
- Source address [32 bit]: source host address.
- Destination address [32 bit]: destination host address.
- Options: it may contain zero, one, or more options. This field is not often used.

The IP address consists of 32 bits and is divided into network parts, the most significant bits, and host parts, the least significant bits. The first indicates the network address, the second indicates the host address, and it is unique for the whole network. Hosts connected to the same network have the same network part, but different host parts.
A network has the following properties:

- Network ID (or subnetwork ID) is the network address, it has the most part set to 0 (e.g. 223.1.1.0).
- Limited Broadcast (255.255.255.255), sends a broadcast packet to all hosts of a certain network, it does not cross over the router and exit from the network.
- Directed Broadcast (223.1.1.255), sends a broadcast packet to hosts on a particular network.
- Loopback (127.0.0.1) used for debugging purposes, it is an address that identifies the localhost.

IP addresses depend on the physical network to which a host is connected. If a host move from two different networks, it has different IP addresses, because the Network ID changes. The IP address is not portable, unlike the MAC address which is unique and always is that because it was established by the customer. In this way, there is the scalability of routing, because, in the router tables (Routing Table), you no longer have to write the whole list of devices, but you can write only the networks.

There are different methods to divide the IP address space:

- Classful addressing: the static division between the network part and the host part defined by some initial bits, Most Significant Bit (MSB).
- Classless addressing: the static division between the network part and the host part. The concept of class is completely removed and there is the use of Network ID and Netmask.

The classful addressing method gets the first bits of the network part and based on these, there are reference classes if the first bit is 0 (MSB), the address is Class A, and the network part ends after the first 8-bits, in this way you can have about 128 different networks. With addresses with 10 (MSB) are Class B addresses, networks up to 16-bit so you have about 16k different networks, and with 110 (MSB) is Class C, network up to 24-bit, with 2M of different networks. This way of dividing address space is not more used today.

The dynamic method does not use classes but uses the standard CIDR (Classless Interdomain Routing) [18]. This method uses a special address called a Network Mask or Netmask, which has all 1-bit in the network part and all 0-bit in the host part. Using the Netmask, and using the bitwise logical AND operator with the address, you get the network ID.
In this standard, to make it more user-friendly, the Netmask is indicated with the prefix length, it is after the address and indicates how many bits the Netmask is long (e.g. 200.23.16.0/23 with Netmask 255.255.254.0). A Network ID, to be valid, must have the bits of the host part to 0, i.e. it must not have a Netmask larger than the network part.

The netmask indicates how many hosts a network can have, for example, an /8 has more hosts than a /24, because the network part is smaller and the host part is larger, also with the use of a netmask a larger network (smaller netmask) can contain more small networks (larger netmask), as it has more addressing spaces, es 200.23.16.0/20 may contain 200.23.16.0/23.

3.1.2 Transport Layer: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

The Transport Layer, as described in [15], is necessary in order to understand, once the destination received the packets, to which application such packets must be delivered. To differentiate the applications, the concept of “port” is introduced. This operation is called demultiplexing because the traffic flow at level 3 arrives for a single host (only destination IP) and must be demultiplexed to applications based on the port number. On the opposite side, there is multiplexing, that is many applications that send traffic and IP has to multiplex it, that is it comes out with a single source IP address. In multiplexing, we focus on the source port and in demultiplexing on the destination port.

The Transport Layer is used to add new functionality, for example, the IP network is a datagram protocol and it is not oriented to the connection, it does not give guarantees. It specifically does not guarantee the delivery of segments, the orderly delivery of segments, or the accuracy of the data contained in the segments. For these reasons, it is possible to guarantee that traffic arrives at its destination by implementing the ACK, which confirms when the packet is delivered correctly. With Transport Layer, there are more functionalities, such as flow control and congestion control. The flow control guarantees that the receiver is not overcharged, and the congestion control guarantees that the network is not overloaded.

The main protocols of this level are Transmission Control Protocol (TCP) [19] and not User Datagram Protocol (UDP) [20]:

- UDP offers a connection-less service, it sends data without worrying about opening or closing connections.
- TCP offers a connection-oriented service, opening and closing connections with the host, and then informing the receiver that you are going to send something.

UDP is an unreliable service, it does not ensure that information sent by one process will be received by the intended process in an accurate manner. Instead, TCP, using flow control, sequence numbers, and acknowledgements, ensures that data is delivered from the sending application to the receiving application. Flow and congestion control can only be used with Transmission Control Protocol (TCP) and not with User Datagram Protocol (UDP).

The transport layer receives segments from the network layer below, its role is to transport these data to the relative application process running on the destination host. For example, if there is a server with four processes when the transport layer receives segments from the network layer, it needs to send the received data to one of these four processes, which are listening on a socket, which is a door through which data arrives to the listening process, and sending it from the process.

In order to properly exchange data between applications, the well-known ports standard has been defined where fixed ports are assigned to specific applications. For example, TCP port 80 is reserved for the HTTP protocol, which will be processed later, so all HTTP servers must be played on port 80. With the source port, there is no need to use a standard, this is because when the receiver reads the port used by the sender, he can use it to respond. For this reason, the source ports are decided randomly. In general, there is no control over the ports used and they can be changed.

User Datagram Protocol (UDP)

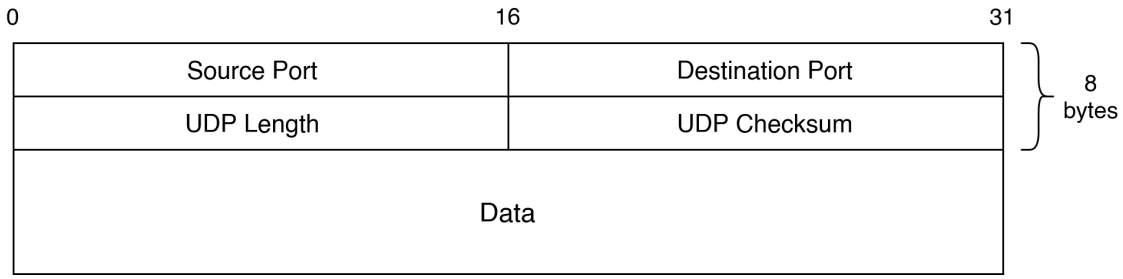


Figure 3.2: It describes the UDP header. At the top is described the number of bits that each field occupies, on the side, it indicates the length in bytes of the selected fields.

UDP does not improve IP best-effort service. The header is small, only 4 bytes, it has Source Port and Destination Port, and it does a checksum on the whole segment. UDP has no ACK, i.e. no guarantees, and it does not control flow/congestion.
UDP is better to use during streaming applications since there are no delays due to retransmission and the flow/congestion control that lowers the transmission rate. In the case of streaming some data can be lost, with a reduction of the quality, but this can be acceptable because it avoids the control of flow and congestion and delays.

Transmission Control Protocol (TCP)

With TCP there is reliability. TCP implements solutions based on window protocols to try to offer reliable services to applications that are at the top level. As described in [15], there are two basic approaches toward pipelined error recovery that can be identified: Go-Back-N and selective repeat.

- Go-Back-N: The sender is allowed to send several packets (when available) without waiting for an acknowledgement, but is limited to a maximum number of unacknowledged packets in the pipeline. In the end, the receiver sends a cumulative ACK, which indicates that all packets have been correctly received at the receiver. When there are N packets, waiting for ACK, there is a single timer, related to the older packet, and when it expires sends the whole window of packages again.
- Selective repeat: The sender can have up to N packets waiting for ACK in the pipeline and the receiver sends an individual ACK for each packet. The receiver keeps a timer for each packet found.

As described in 3.1, TCP is connection-oriented because, before one application process can send data to another, the two processes must first "handshake," to communicate certain preliminary segments to each other in order to define the parameters for the data transfer.
Many TCP state variables connected with the TCP connection will be initialized as part of the TCP connection establishment process on both sides of the connection.

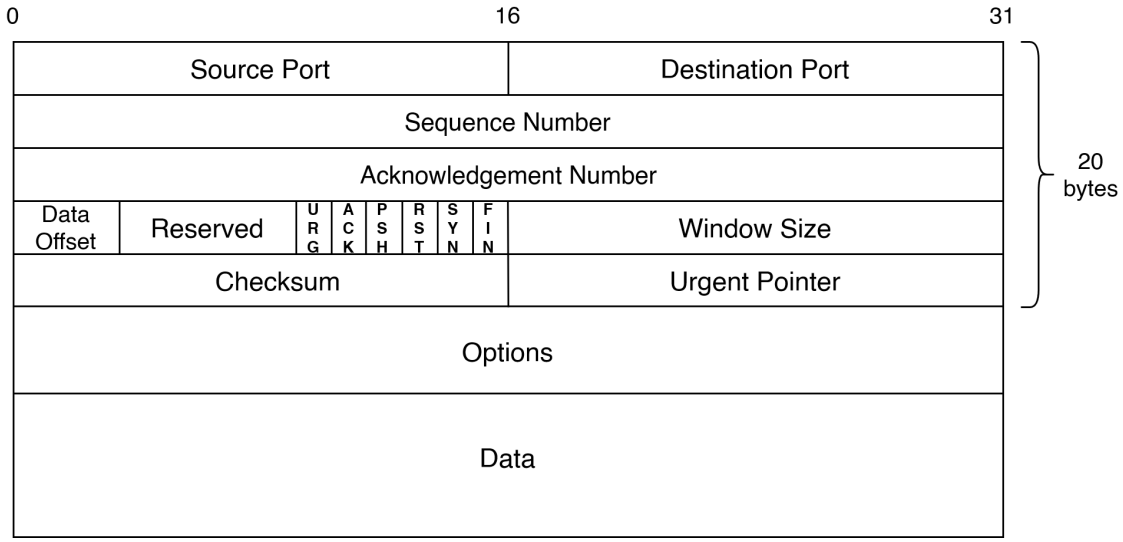


Figure 3.3: It describes the TCP header. At the top is represented the number of bits that each field occupies, on the side, it indicates the length in bytes of the selected fields.

As described in Figure 3.3, the TCP header consists of 20 bytes, without the Options field. The header has Source and Destination Port numbers for multiplexing and demultiplexing data.

- The Sequence and ACK Number [32 bit] are used by the TCP sender and receiver to count the segments and to find the received segments.
- Data Offset [4 bit] is the TCP header length field. The TCP header can be of variable length based on the TCP options field.

- Flag Field [6 bit] has 6 bits, each for a specific flag:
  - URG indicates that the sending-side upper-layer entity has designated certain data in this segment as “urgent”.
  - ACK indicates that the segment includes an acknowledgement for a successfully received segment.
  - PSH indicates that the data is sent to the upper layer immediately by the receiver.
  - RST, SYN and FIN are used to open and close the connection.
- Window Size [16 bit] is used for flow control. This parameter represents the number of bytes that the receiver can accept.
- Checksum is equivalent to that of UDP, and it is calculated on the header and payload.
- Options are used for window scaling factor in high-speed networks or when a sender and receiver agree on the Maximum Segment Size (MSS).

TCP provides a connection-oriented service with reliable data transfer over unreliable IP protocol. This can be done with: a pipeline, cumulative Acknowledge and a Timer.

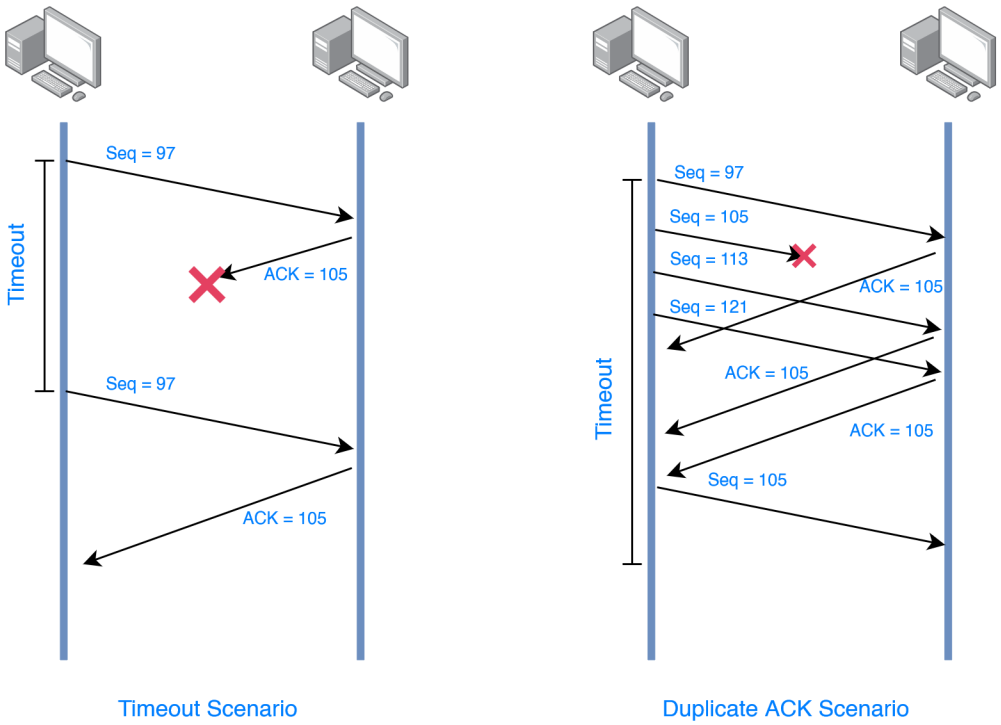


Figure 3.4: It describes what happens when an ACK is lost, Figure on the left, then the timer expires and the sender sends the packet back, while in the figure on the right the sender sends several packets, and one of them is lost, in this case, the recipient returns numerous ACK with the number of bytes of the packet missing.

As described in Figure 3.4, in general retransmission of a packet can be triggered by two events:

1. Timer timeout, TCP retransmits a packet, possibly because it was lost in the channel.

2. Duplicate ACK, in TCP ACK, is cumulative, but it does not say what is the last byte received, but it says what is the next byte that expected to receive. The receipt of three duplicate ACKs for a given segment triggers the retransmission of that segment because it is a sign that there was a problem and the related package is sent again.

Referred to Figure 3.4, a TCP transmitter performs the following steps:

1. It receives the byte stream from the application layer and it builds the segments using sequence numbers.
2. If there is not already an active timer, it starts one. The timer refers to the oldest package with no ACK received yet.
3. If the timer timeout only the segment that caused the timeout.
4. When receiving the ACK the timer relative is stopped, but there is the need to start a new timer if there are still ACK to be received for other segments.

In general, the TCP timer is longer than the Round Trip Time (RTT), which is twice the propagation time, that is, the time from when a segment is sent until an acknowledgement is received. The sample RTT, also called SampleRTT, for a segment is the interval of time between when the segment is sent and when an acknowledgement (ACK) for the segment is received.

Over time, RTT changes according to network congestion. It is therefore difficult to estimate the RTT to initialize the timeout. For this purpose, the EstimatedRTT is used:

$$EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT$$

where α has as recommended value 0,125 or 1.
EstimatedRTT is a weighted average of SampleRTT values, and this weighted average gives more weight to recent samples than to old ones, because recent samples better reflect current network congestion.

Flow Control

Flow control is the ability of the transmitter to avoid overloading the receiver.

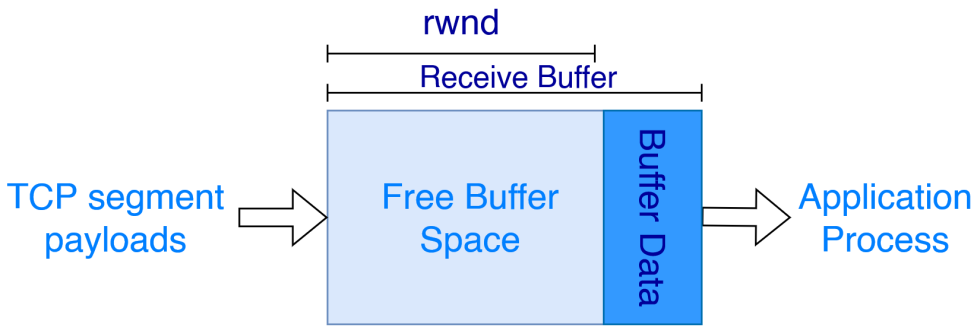


Figure 3.5: It describes the receive buffer. On the left comes the correct frames and verified by TCP, from the right instead there are applications that take this data from the buffer. When the data arrives, the buffer fills and decreases the rwnd, i.e. receiver window.

To summarize, when the TCP connection receives bytes, it does not send them directly to the application, but they are in a buffer and then it is the application that takes the data from the buffer, not necessarily at the instant, the data arrives and then uses them. If the application struggles to read network traffic that comes in this buffer, the buffer fills up until overflow (memory in the buffer ends) and the received bytes have no more space. Flow control avoids overflowing the buffer.

As described in Figure 3.5, the buffer of the receiver is composed of a part with the received data waiting to be read (Buffer Data) and an empty part waiting for the new segments (Free Buffer Space), this last empty part is the receiving window. The reception window gets smaller as the buffer is full.
It is necessary to provide the rwnd value of the transmitter in order to establish flow control. In this way, the buffer will never overflow if the transmitter sets the value of its transmission window to rwnd. The TCP header includes a field for the rwnd value, called "Window Size".

3.1.3 Transport Layer Security (TLS)

TLS is the most adopted network security protocol nowadays. Initially, it was called SSL (Secure Socket Layer) [21] [22], but this term shouldn't be used anymore because it refers to a protocol that was discontinued and had security vulnerabilities. In January 1999, TLS was standardised [23].
TLS constructs a secure transport channel on top of Transport Layer 4, also known as session-level security, for which it is also referred to as a layer 4.5 protocol.
TLS is a security protocol and it has the following Security Proprieties:

- Peer authentication establishes when a channel is opened. If there is single peer authentication, only the server is verified, if there is mutual peer authentication, both the server and client are verified. It is based on asymmetric challenge-response authentication, which means that should be proof of possession of the private key. For the server, this is indirect proof because the server, without its private key, can't establish a secure channel. On the other hand, there is an explicit signature for the client.
  If peer authentication fails, the channel is not opened. This is important because if an attacker can't make a connection to us, he can not perform an attack. That is why it is so important to do authentication at the network level and not at the application level. The sooner the authentication is performed in the network stack, the more you can block an attack.

- Message confidentiality means that the content of each record sent over TLS can be encrypted if needed. It depends on the version of the protocol, but taking into account that encryption takes a long time, especially when there are a lot of data. In some cases, the data is not private, so message confidentiality is not needed.

- Message integrity and authentication for each message sent over the TLS channel. In this case, authentication means that data sent over the channel really came from the counterpart; it also proves that the data has not been changed. Integrity only tells you if data has been changed or not, but does not prevent changes.

- Protection against replay and filtering attacks, where a valid message is taken and re-injected again. TLS protects against replay attacks because if an old message is sent again, it is noticed by implicit record numbering. This means that each

message sent over TLS has a numbered record, that is not inside the packet, but it is handled by the top-layer application which doesn't lose data because it sends and receives data in the same order. This protects against replay attacks, if a message is received twice, authentication fails. In the same way, it protects against message cancellation.

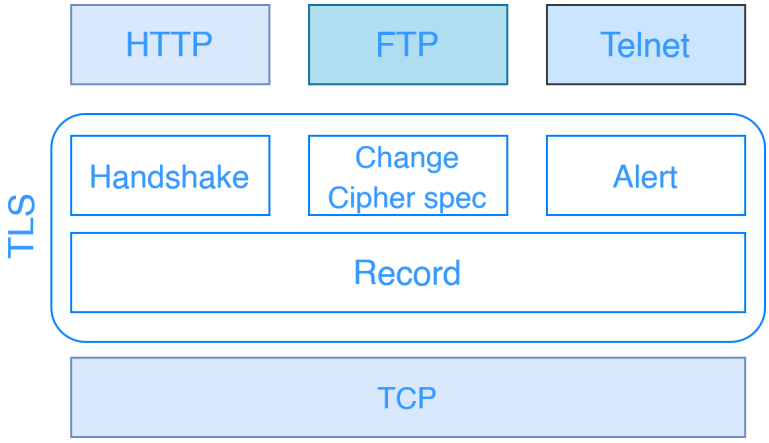


Figure 3.6: It describes the TLS architecture, which is located over TCP and below the application layer (HTTP, FTP, Telnet). The TLS architecture consists of a handshake protocol, change Cipher specification protocol, alert protocol and record protocol.

As already mentioned above, TLS is located above the TCP transport layer and below the application layer. As shown in Figure 3.6, the architecture of TLS consists of:

- Handshake Protocol is the protocol that is run at the beginning when the TLS channel is created. It is the most vulnerable part because it is where an unprotected TCP channel becomes a secured TCP channel.
- Change cypher spec Protocol used for changing keys or algorithms without closing the channel. This is important to avoid cryptoanalysis from the attacker since, during the handshake phase, a lot of data using the same algorithm and key are exchanged.
- Alert protocol is the protocol utilized each time there is a problem. The error means that there was an attack in between, so the security manager is alerted.
- Record protocol is the protocol that encapsulates the data being transmitted. The basic data transmitted is the record.

TLS versions

TLS 1.0 [23] was the first version of Transport Layer Security that was standardized by the Internet Engineering Task Force (IETF). It is 99.9% the same as SSL-3, but it adds more focus on digest and asymmetric cryptography algorithms.

TLS 1.1 [24] was made to protect against CBC attacks, where some parts of the message could be guessed. To avoid these kinds of attacks, the implicit IV has been changed to an explicit IV, and errors in padding are now reported as bad_record_MAC instead of decryption failed. This is important because if there is an error, the attacker can not get any information.

TLS 1.1 also added the IANA registry, which defines protocol parameters. If a session ends for a network problem, the session can resume again, but, if the channel was closed because of an attack (for example, an alert for a bad MAC), the session can't be started again because the attacker could have changed the session.

TLS 1.2 [25] is the most popular version right now, even though TLS 1.3 [26] is the most recent. The cypher suite now says what pseudo-random function (PRF) is needed to generate the client and server random. The hash algorithm used in the MAC can be negotiated, SHA-256 is used by default. The protocol extensions and the AES cypher suite are built into TLS 1.2.

3.1.4 Cyber-Attacks

This subsection presents the cyber attacks that are used in the proposed model.

Denial-of-service (DoS)

Denial-of-Service is an attack technique that attempts to overload a host so that its service cannot be provided.
For example, you could send a lot of emails to keep the server busy until you get the message "The destination mailbox is full." This attack is called "Mail Bombing" [27], and it is a DoS attack which uses mail. There is also a Ping flooding attack, also known as "ping bombing", in which it used a lot of ICMP echo requests, with the biggest amount of bites (64 Kbytes) and without the timer, then the host will be too busy answering all of these packets and it does not perform other tasks.
Another important DoS attack is the SYN flood [28], which uses the TCP of the transport layer, described in the section above. An attacker starts a connection to a server quickly but doesn't finish it, then the server has to spend resources waiting for half-opened connections and it can not perform other tasks.

In other words, Denial-of-Service is comparable to an increase in the number of users of a service, the system is flooded with requests, and it goes down because its resources are not sufficient to maintain its service active. Monitoring and scaling up can reduce the consequences, but the system should have an alert system that sends an alert to the system manager when a certain resource threshold has been exceeded.

Distributed Denial-of-Service (DDoS)

DDoS is similar to DoS, but multiplied by the number of attackers who are attacking at the same time [29].
A number of compromised devices form a Botnet, and the attacker installs DoS software on these devices. Each of these nodes is referred to as a daemon, a zombie, or a malbot. Typically, a C&C (Command and Control) infrastructure, either client-server or peer-to-peer, is used to manage a network of robots (slaves) that are managed by a master. The communication between zombies and master is either encrypted, and this type of traffic can be hidden to avoid future investigations, as ICMP messages are commonly present in typical traffic.

The more demons there are, the more the effect of the attack can be amplified.
The use of a "reflector" is another way to improve an attack, which is a possibly legitimate third-party component that increases the traffic sent to a victim. This method has two

main benefits: it hides the attacker's identities and makes the attack more powerful by using an "amplification factor".

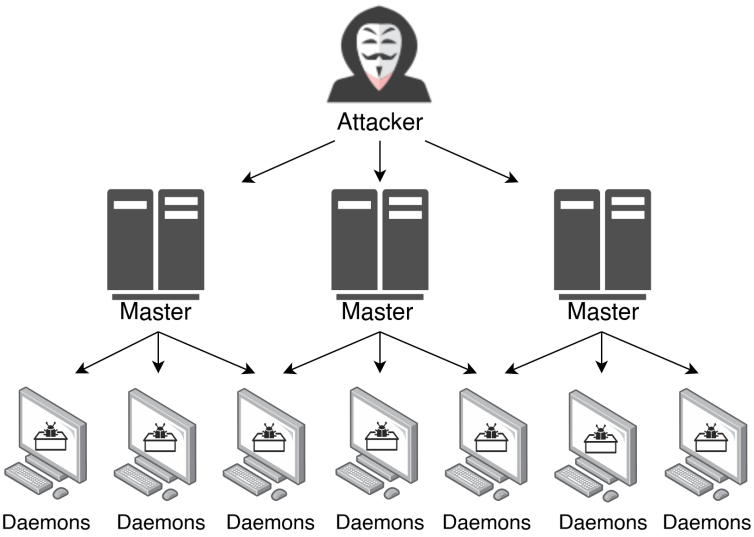


Figure 3.7: Description of how the botnet is controlled by the attacker. At the top, there is the attacker who controls Master nodes which control daemons.

With DDoS, there is an attack and a victim. Masters and daemons are part of the botnet. As described in Figure 3.7, the attacker builds a network of daemons and then picks some nodes to be botnet masters. When the attacker wants to start a DDoS attack, he sends the victim's IP address to master nodes that propagate information to all daemons, which start sending traffic to the victim server. The attacker then disconnects from the network to avoid being tracked after sending the victim's IP address to attack to the master nodes. The other masters propagate the victim IP over the botnet and they coordinate the work of the demons, which must simultaneously execute the attack. Masters do not directly participate in the attack; instead, they attempt to remain as hidden as possible to prevent being found during an investigation. The victim is flooded with requests and is killed by the huge amplification factors that a daemon might generate.

Infiltration

Infiltration is a sophisticated attack where the system is penetrated using certain complex techniques and it is continuously monitored from an external Command and Control system (C&C). It is often performed by a group of advanced attackers that are well-funded by an organization or government [30].

As described in [30], this attack is divided into 5 stages:

1. Reconnaissance: in order to increase their success rate, attackers try gathering all the information they need about the organization's assets.

2. Establish Foothold: Based on what they learned in the first step, the attackers try to use different techniques: exploit known vulnerabilities in the victim, use "spare-phishing", which is a type of phishing directed at a company, with malware.
   After sending emails with malicious files or links to websites to malicious software, attackers wait that the victim opens the malware in the organization's network,

which would let them into the organization's system. After the attackers get the organization's system control and infiltrate the targeted network, attackers create a Command and Control (C&C) communication channel after they gain control of the organization's system. Attackers establish a long connection to victim devices for monitoring and stealing sensitive data.

3. Lateral Movement/Stay Undetected: once the attacker gains access to the victim system, the attacker can spread to additional systems inside the internal network of the target, always remaining as hidden as possible.
   The attacker utilizes a variety of techniques to get access to additional hosts and sensitive resources from a compromised system. Typically, this phase involves privilege escalation and, at times, the acquisition of sensitive data via key loggers. The attacker can also install new backdoors on many systems, utilizing VPN credentials, and log into web portals.

4. Exfiltration/Impediment: the attackers export the stolen information to their command and control server. This can be done, since the majority of Intrusion Detection and Intrusion Prevention Systems, described in the next section, only perform ingress filtering, their data exfiltration may go undetected.

5. Post-Exfiltration/Post-Impediment: the objective of an APT attack is not to perform harmful actions, but it is to collect as sensitive data as possible. The attackers could choose to end the attack once it is determined that the data retrieved are accurate, or it could remain active for as long as the attackers continue to collect data. In each situation, the attackers must hide their traces, if there is no need for further exfiltration, any tools installed during the attack or logs that could provide solid proof are uninstalled.

Heartbleed

Heartbleed is one of the attacks against TLS. It is a critical bug in OpenSSL heartbeat extension of TLS/DTLS [31], used to read portions of the victim server memory.

Handshake is time-consuming and, because of the numerous closing and reopening of the channel, even if the session is resumed, for a better performance this extension maintains the channel open even if no data is being sent. It is a message sent to the server via 'heartbeats' that prevents the channel from being closed.

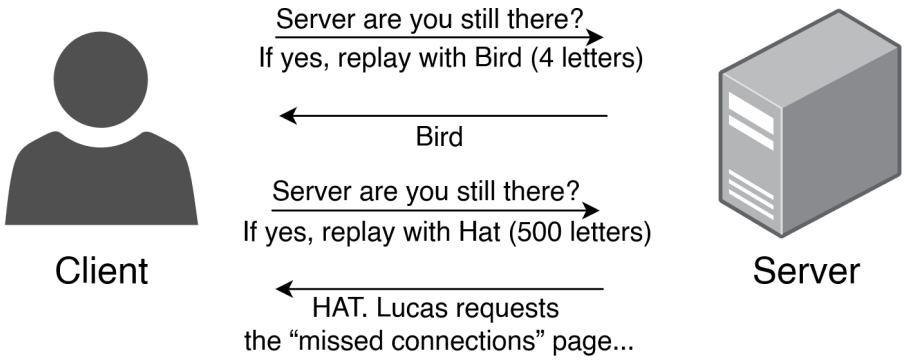


Figure 3.8: Description of how the Heartbleed vulnerability works.

Figure 3.8 shows how the Heartbleed vulnerability works. The client sends a HeartbeatRequest, in which it sends the word "Bird" of 4 letters, and the server responds with

HeartbeatResponse that contains the same word sent with HeartbeatRequest. At this point, the client decides to send the word "Hat" but enters the value of 500 letters. The server then returns not only the word Hat but also data that was present in its memory.

This extension provides two messages: HeartbeatRequest and HeartbeatResponse. A HeartbeatRequest message can be sent at any time during the TLS connection. Whenever a HeartbeatRequest message is received, a HeartbeatResponse message is sent in response. The receiver, so the server, must send back with HeartbeatResponse the same message contained in the HeartbeatRequest payload, otherwise, if the message does not contain the same payload, the message is discarded.

This extension caused a vulnerability, which is named CVE-2014-0160, that exploited a buffer-overflow in OpenSSL (from versions 1.0.1 to 1.0.1f)[1].

An attacker can send a HearthbeatRequest, where he can set the length of the message, which is greater than the real message contained in the payload. The attacker is able to perform a buffer overflow, and the server returns, with HeartbeatResponse, more characters than expected in HearthbeatRequest, without checking the real length of the message.

By taking advantage of this vulnerability, the TLS server sends back up to 64KB more data than was asked in the HearthbeatRequest. Attackers can get sensitive information that is stored in RAM.

Bruteforce

A bruteforce attack is used to illegally obtain username and password pairs by trying all existing combinations to access network services. The attack is executed as many times as possible until it is successful [32]. It is a trial and error strategy to crack login credentials and encryption keys, for that it is called "bruteforce", for the attacker's aggressive attempts.

There are several bruteforce attack techniques:

- Dictionary Attacks, in which the attacker compares the user password with a word list, so a dictionary containing the most common password. This form of attack is often time-consuming [33].

- Rainbow tables are essentially enormous tables containing hash values for the most common passwords (that are not salted, which have not been randomly sequenced), and these tables are used to discover hashed passwords. The authentication login uses hash functions, which are one-way functions, meaning that the hash cannot be reversed to recover the original plaintext. When a user wants to access to the system, a hash is computed based on the password he has inserted; if the hash value matches, the user gains access to the system.
  Rainbow tables are utilized to crack passwords in a shorter amount of time than the simple bruteforce with a dictionary attack, however, they require a large quantity of storage space. It is the most effective way of password cracking [34].

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160

Botnet Attacks

A Bot is a robot that generates non-human traffic to a website or mobile application. Per se, a Bot is not malicious, because that can be a search engine or assistant, but in some cases, a Bot can be programmed to perform DDoS or Bruteforce attacks or also to steal credentials. In some cases this bot has so sophisticated that can act as a human and write messages or emails to the victim, causing him phishing attacks.

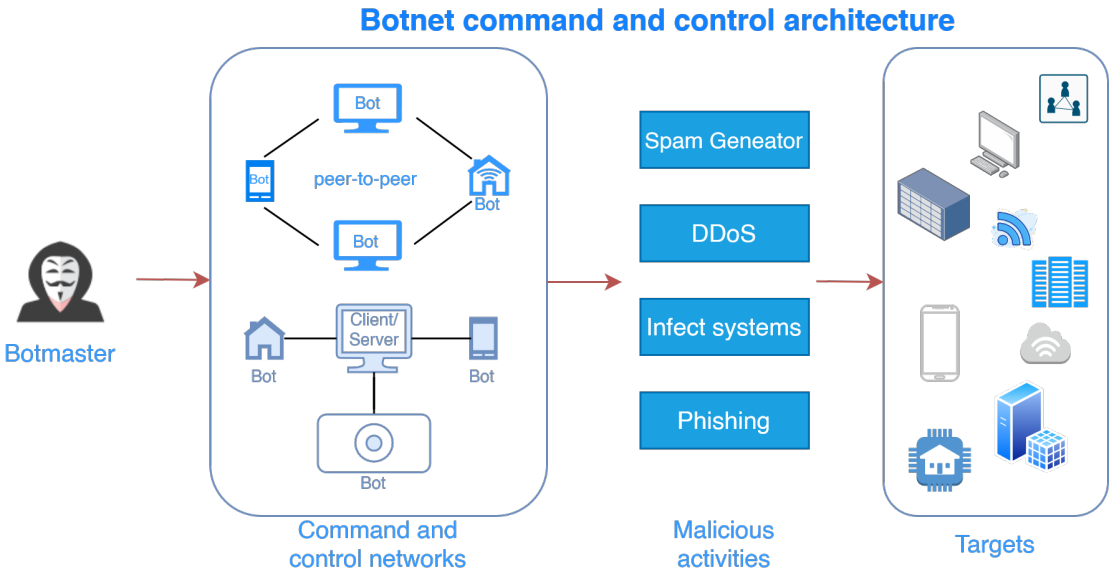


Figure 3.9: It describes the bots used by the attacker and the types of attacks that they can perform.

The malicious bot can be anything device that is connected to the internet and that is infected by the attacker. The attacker, in Figure 3.9 is the Botmaster, controls the botnet and sends them instructions as to what kind of attacks can generate.

It is estimated that more than 40 per cent of all Internet traffic consists of bot traffic. This is why so many firms are searching for methods to control the amount of bot traffic on their websites [2].

There are different kinds of bots:

- Scraper Bots are applications that extract information from a web application and export the data in the desired format (JSON, XML, HTML, etc.). These type of bots produces high traffic to a website and they can be used to discover different path and hide resources inside your server. Their aim is to get all the information possible about the server and use it for malicious purposes.
- Spam Bots are software that mimics user behaviour. These are the most dangerous bots because they are so sophisticated that can generate not only traffic and perform a DDoS attack, as previously said, but can create and send messages over social platforms and spread misinformation. Research "find that up to 66% of data sets of known bots are discussing COVID-19", and "the proliferation of COVID-19 (mis)information by bots, coupled with human susceptibility to believing and sharing misinformation, may well impact the course of the pandemic" [35].

[2] https://www.cloudflare.com/it-it/learning/bots/what-is-bot-traffic/

3.1.5 Intrusion Detection System (IDS) and Intrusion Prevention System (IPS)

This section describes what an IDS is, the different approaches used (signature-based and anomaly-based) and the different types (HIDS and NIDS). Then the advantages and disadvantages of signature-based and anomaly-based approaches are also discussed with some examples.

The concept of IPS will also be discussed in the end.

Intrusion Detection System (IDS)

Intrusion Detection System (IDS) is the process of monitoring what occurs in a computer system or network and analyzing it to look for intrusions. Depending on its features, it can be improved to find unauthorized access, malware or DoS-DDoS attacks, also the system must be able to recognize and differentiate an attack from normal network traffic [36].
Based on this definition, an Intrusion Detection System (IDS) is just the tool, whether it's software or hardware, that finds intrusions in a host or network.

IDSs are mostly used to find potential malicious events, record information about them, and alert security administrators about what they found. An IDS can base its behaviour on two strategies:

- Signature-based detection is the process of looking for potential attacks by comparing signatures with events that have already happened [36]. In practice, for each "event", it generates a signature, and then compares that signature with a database that contains all the signatures of malicious events, if there is a match, the system sends an alert to the security manager.
  Signature-based detection is the easiest way to find malicious events because there is just a comparison of signatures. Signature-based IDSs can detect only known attacks, that are in the database, but for new attacks or different versions of known attacks (atypical attacks), they do not recognize them at all.

- Anomaly-based detection is the process of recognizing the malicious traffic from normal traffic, using statistical data [36]. A system for finding malicious traffic has rules that describe how users, hosts, network connections, or applications normal act. During a period of time, these rules were made by looking at how normal things were done.
  If, for example, the rule is the average number of packets exchanged on a web server and the IDS, using statistical methods, determines that the number of packets exchanged is much higher than expected, then an alert is sent to the system administrator. Rules can be made for many kinds of behaviour, like how many emails a user sends, and how many times a user tries to log in and fail.
  The rules used for anomaly-based detection can be static or dynamic. Static rules don't change unless the system administrator changes them in the IDS, dynamic rules are updated as new things happen. Static rules get old over time, so it needs to be updated.
  The most important advantage of anomaly-based detection methods is that they can find new types of attacks.

There are two types of IDS:

- Host-based IDS (HIDS), which examines logs created by the operating system, a service, or an application on a single host. Additionally, it monitored OS tools.
- Network-based IDS (NIDS), which monitors the traffic on the network.

Using machine and deep learning, as detailed in Chapter 3.2, is intended to automate the analysis process without human intervention. These techniques can utilize supervised, unsupervised, and semi-supervised learning approaches. This means that the security manager needs to check the alarms that the IDS send, investigate the problem, and find out if there is an attack or if it is just a false positive.

The main advantage of IDS Anomaly-based is that is not dependent on a database, which means that it behaves using statistical analysis to find known and unknown attacks, rather than the IDS signature-based. Let's say for example that there is a new type of attack, the signature-based does not recognize it because there is no signature for that attack, as it has never been discovered before, instead the anomaly-based because that traffic was never discovered before, then that is not benign but an anomaly.

If we have an IDS anomaly-based that is trained to recognize all the benign traffic under a threshold, then all the traffic that is over this threshold is considered malicious, and a possible attack. This is an example of the use of an unsupervised learning model to build an IDS. On the other hand, when the anomaly-based model is inserted in a network, you need to train the model to recognize the benign traffic generated from that network or it will produce a lot of false positives. If the model is trained on a small dataset, that does not contain all the possible benign traffic, then if there is a flow that is benign, but has never been seen by the model, that is considered malicious traffic.

Intrusion Prevention System (IPS)

An intrusion Prevention System (IPS) is a system used to automate and improve the response to attacks. It combines the Intrusion Detection System with a dynamic firewall [37]. The IDS sends alarms about a certain type of traffic, and when the dynamic firewall receives the alarm, it is able to block all traffic of that type or isolate a node.

This is useful because the system can detect an attack and directly block it, on the other hand, if the system is not accurate then IPS may block legitimate traffic or may consider malicious traffic as normal traffic.
IPS is composed of different protection systems, and frequently they are integrated into a single product known as IDPS [38].

3.2 Basic Concepts of Data Science

This section describes the basic concepts of Data Science.

Artificial intelligence (AI) is the machine simulation of human intelligence, mainly by an algorithm called also as "model". As described in [39], the term Artificial Intelligence is used when a computer simulates "cognitive" functions associated with human minds, such as "learning" and "problem-solving". Understanding spoken language, excelling at strategic game systems like chess, driving autonomous vehicles, using intelligent routing in content delivery networks, simulating military operations, and decoding complex data.

In Artificial Intelligence, there is a subset which is Machine Learning. Machine learning is the study and creation of algorithms that can learn from and make predictions on

data [40]. These algorithms avoid purely static program instructions by making data-driven predictions by creating a model from the inputs, which is called a dataset.

Deeper in Machine Learning, there is another subset called "Deep Learning", which is the study of Artificial Neural Networks (ANN) with several hidden layers, made of neurons, in order to describe data abstractions and build computer models [41].

This chapter describes the methods for preprocessing a dataset and how to evaluate a model using a Confusion Matrix. In addition, it discusses the main Machine Learning and Deep Learning strategies utilized in the building of different models.

3.2.1 Dataset and Preprocessing

A dataset is a collection of data with a specific structure, which is defined with rows and columns. Each row of the dataset is called a record (or example) and it contains the same structure as the dataset. Each column is called a feature (or attribute or input), this can be numeric, categorical, text, date-time or Boolean.
A particular feature is a label, which indicates the "prediction" based on all the other features [42].

Training, Validation and Testing

As described in [43], a dataset is splitted in Training Set, Validation Set and Test Set:

- Training Set is a collection of instances used for learning, that is, to fit the classifier.
- Validation Set is a collection of examples used to tune the parameters (hyperparameters) of a classifier, such as selecting the number of hidden layers in a neural network.
- Test Set is a collection of instances used to evaluate the performance of a classifier.

To sum up, the Training Set is used to train the model, which means the model learns from the data and tries to establish a relationship between the features and the labels.
The Validation Set, as already said, is used to tune the hyperparameters of the model. When the model uses certain data, but never "learns" from them, the Validation Set can be used to find the hyperparameters at a higher level. This dataset is useful during the model's development phase.
The Test Set provides a way to evaluate the final model. It is only utilized once the training of the model is complete (After using the Train and Validation sets). Typically, the model should be evaluated on samples that are not used to train or validate the model.

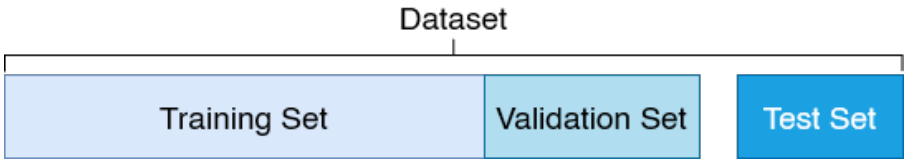


Figure 3.10: It shows the split of the Dataset into Training Set, Validation Set, and Test set.

As described in Figure 3.10, data are divided according to a split ratio that is highly determined by the type of model being constructed and the dataset itself.
If the dataset and model require extensive training, a larger portion of the data for the training set is used and a smaller part is used for validation and test set.

Data Preprocessing

As already said, the dataset has different records that contain different features, each feature has a different type, and it can be numeric or categorical, which is a non-numeric value. When all of the features are categorical, this must go through a "Data Preprocessing Phase" because Machine Learning models do not understand the "text", but only numeric values. Moreover, also the numeric features can also go through a preprocessing phase, to make these values easier to understand during the training phase.

Data Preprocessing is the step in which the data is encoded or otherwise modified so that it can be easy to understand by the model. In other words, the algorithm can analyze and learn from data quickly [44].

Normalization

Data normalization is a fundamental component of data mining. It transforms or translates the data into another format that makes easier the data processing. Normalization reduces the complexity of the data and improves the performance of model training.

Several data mining normalization approaches are utilized extensively for data transformation:

- Min-Max Normalization or Min-Max Scaling performs a linear transformation on the original data by scaling the range of features to the interval [0, 1] or [min(feature_i), max(feature_i)]. The generic formula used for scaling to a new min-max [new_min, new_max] is as follows:

$$v' = \frac{v - min(v)}{max(v) - min(v)}(new\_max - new\_min) + new\_min$$

  Where v is the value assumed by each feature.

- Standardization or Z-score Normalization is the transformation of data by the values to a common scale in which the mean equals zero and standard deviation equals to one.

$$v' = \frac{v - average(v)}{\sigma_i}$$

  The above formula describes how to determine the Z-score with the mean ($average(v)$) and standard deviation of the distribution for each feature σ_i. The mean is subtracted from each value and then, it divides the values for standard deviation.

As described in Chapter 4, Min-Max Scaling and Z-score Normalization are managed by specific functions from the scikit-learn library [45], respectively MinMaxScaler[3] e StandardScaler[4].

[3] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

3.2.2 Confusion Matrix and other Metrics

For the evaluation of the performance of the models, the "Confusion Matrix" is used, it is a table with four different combinations of predicted and real values.

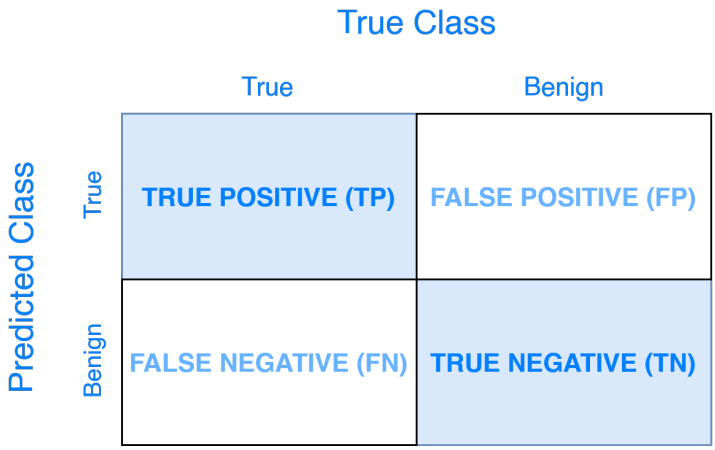


Figure 3.11: Matrix Confusion. The provided classes are those predicted by the model, while the actual classes are the correct ones.

From the confusion matrix Figure ?? is described, if the prediction of a class is correct, for example, the model predicts that the traffic flow is a DoS attack, then it is a True Positive (TP), whereas if the prediction is wrong, it is a False Positive (FP). If the model predicts that traffic flow is not a DoS attack and the forecast is accurate, it is a True Negative (TN); otherwise, it is a False Negative (FN). Then, there are four different indicators:

- **True Positive (TP)**: describes the number of cases predicted by the model as positive for that class, and they are actually positive.
- **False Positive (FP)**: describes the number of cases predicted by the model as positive for that class, but they are not positive.
- **True Negative (TN)**: describes the number of cases predicted by the model as negative for that class and they are actually negative.
- **False Negative (FN)**: describes the number of cases predicted by the model as negative for that class, but they are not negative.

For example, with a model that is trained to recognize benign traffic from malicious traffic, then assign classes "benign" and "malicious". It creates the confusion matrix for the class "malicious", described in the Figure 3.12:

- If the model correctly predicts the "malicious" class, that is a True Positive (TP).
- If the model predicts the class "benign" to traffic and it is actually traffic "benign", then it is a True Negative (TN).
- If the model predicts the class "malicious" to the traffic but actuality it is "benign", then this is a False Positive (FP).

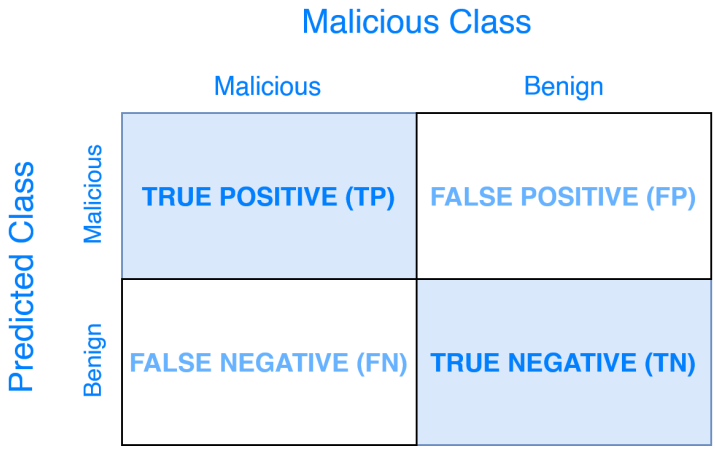


Figure 3.12: Matrix Confusion for the Malicious class. It describes the performance of a model, trained to understand the malicious flow from benign form.

- If the model predicts a class "benign" to traffic, but in actuality it is "malicious", then it is a False Negative (FN).

The confusion matrix is very useful for measuring Accuracy, Balanced Accuracy, Precision, Recall, F1-Score and Area Under the Curve (AUC), as described in [46]:

- **Accuracy** the total number of correctly predicted cases, over the total number of cases.
$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$
- **Balanced Accuracy** is it based on the $sensitivity = \frac{TP}{TP+FN}$, also known as True Positive Rate (TPR), on the $specificity = \frac{TN}{FP+TN}$, also known as True Negative Rate (TNR).
$$Balanced\ Accuracy = \frac{sensitivity + specificity}{2}$$
- **Precision** the number of correctly predicted positive cases, True Positive (TP), of the total number of cases classified by the model as positive, False Positive (FP) and True Positive (TP). Precision is a measure of how many positive predictions are correct, a high precision refers to a low False Positive (FP) number.
$$Precision = \frac{TP}{TP + FP}$$
- **Recall** the number of cases correctly predicted, True Positive (TP), over the number of cases predicted as positive, True Positive (TP), and False Negative (FN).
$$Recall = \frac{TP}{TP + FN}$$
- **F1-Score** combines Recall and Precision, considering both False Positive (FP) and False Negative (FN). A high F1-Score is only achieved if both Recall and Accuracy are high.
$$F1 - Score = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

- **Area Under the Curve (AUC)** is the ability of the model to distinguish the positive classes from the negative ones, that is to distinguish the True Positive (TP) from the True Negative (TN), if this does not happen they generate False Positive (FP) and False Negative (FN). The higher the AUC, the better the performance of the model is.

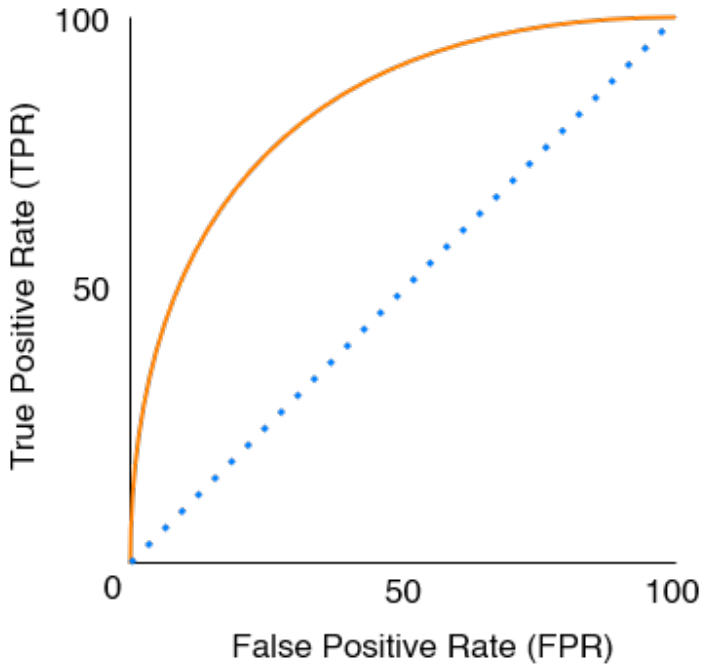


Figure 3.13: The Receiver Operating Characteristics (ROC) curve is a graph, based on True Positive Rate (TPR) and False Positive Rate (FPR), that is used to evaluate the performance of a classifier.

The Receiver Operating Characteristics (ROC) curve is drawn to calculate the AUC. For each model prediction is calculated True Positive Rate (TPR)

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

and False Positive Rate (FPR)

$$FPR = \frac{FP}{FP + TN}$$

These values are updated for each model prediction and they are marked on the chart, Figure 3.13, to build the ROC. After generating the ROC, the Area Under the Curve is calculated measuring the area under the entire ROC curve.

The closer the curve is near the top left corner, the more the model is performing and can distinguish positive from negative classes, in this case, the AUC is higher (> 0.5) and the model has better performances.

If the curve is closer to the dotted line, the model predictions are random because it can not distinguishes the positive from negative classes, this happens with an AUC equal to 0.5.

If the curve is under the dotted line and closer to the bottom right corner, the model reverses the positive classes with the negative one, this happens with a low AUC (< 0.5).

In general, if the curve is under or is equal to the dotted line, there is a low performance of the model and it generates a smaller AUC ($<= 0.5$). Instead, if it is over the dotted line, the AUC is higher (> 0.5) and the model has better performance.

Micro and Macro

Macro-average computes the metric independently for each class and then takes the average, considering all classes equally. Instead, micro-average combines the contributions of all classes to calculate the average metric [47].

For example, consider the following table 3.1:

Class	TP	TN	FP	FN	Accuracy
A	20	15	2	1	0.92
B	105	10	85	67	0.43
C	2	5	15	7	0.24

Table 3.1: Accuracy calculates for each class.

Macro-average is the average of the all Accuracy of the different classes:

$$Macro - average = \frac{Accuracy_A + Accuracy_B + Accuracy_C}{3} = \frac{0.92 + 0.43 + 0.24}{3} = 0.53$$

Micro Average, for accuracy, is the sum of all true classes (TP and TN) divides by the totals elements of all the classes:

$$Micro - average = \frac{TP_A + TN_A + TP_B + TN_B + TP_C + TN_C}{All\_results\_A + All\_results\_B + All\_results\_C} =$$

$$= \frac{20 + 15 + 105 + 10 + 2 + 5}{20 + 15 + 2 + 1 + 105 + 10 + 85 + 67 + 2 + 5 + 15 + 7} = 0.47$$

The first distinguishing characteristic is that the macro-average is greater than the micro-average. This is done because, in the macro-average, classes A and B contribute to increasing the accuracy, in an effort to reduce the poor performance of class C. However, this is an inflated metric, as a large number of examples are incorrectly classified.
Micro-average, instead, considered the number of examples and the contribution of each class, for that reason it is recommended to use with imbalance dataset.

3.2.3 Machine Learning (ML)

As already said and described in [47], Machine learning is the study and creation of algorithms that can learn from and make predictions on data [40]. There are two types of learning: Supervised and Unsupervised Learning.
The main distinction is that the first uses data labelled to predict results, while the second one uses unlabelled data, and tries to classify each record based on similar feature values.

Supervised Learning

Using a dataset containing labelled data, Supervised Learning is an "automatic" learning technique. This data is intended to instruct or "supervise" algorithms in order to correctly classify the data.
Using analysis of the dataset and output labels, the model can improve over time to achieve better accuracy. Using techniques such as Classification and Regression, supervised learning predicts the label values of new unlabeled data based on patterns.

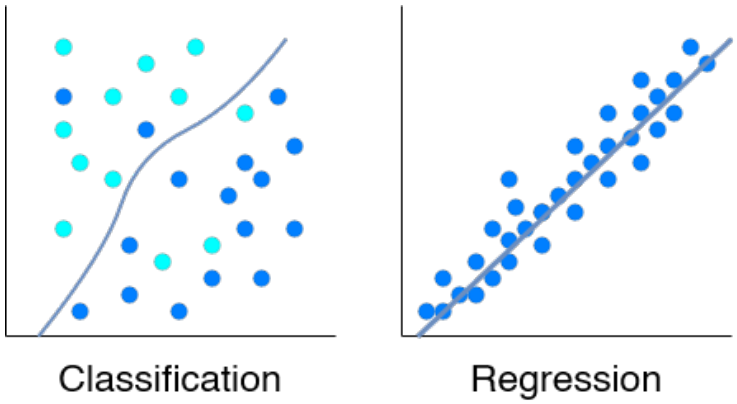


Figure 3.14: Difference between Classification and Regression.

As described in Figure 3.14, supervised learning is divided into two categories:

- Classification: Using an algorithm, data are sorted into various groups based on the label. For example, in Figure 3.14, A line divides the data into two groups based on where the majority of the same labels are located. In the real world, algorithms of this type can be used to identify spam emails from normal ones. Decision Tree, Support Vector Machines, Random Forests, and linear classifiers are typical supervised learning methods.

- Regression: A technique for supervised learning that applies an algorithm to comprehend the relationship between dependent and independent variables.
  Regression models are helpful for making predictions based on previously collected data, such as predicting future profits or the stock market.
  Common regression methods include polynomial regression, Logistic regression, and linear regression.

Unsupervised Learning

Non-supervisionate or unsupervised learning analyzes and groups unlabelled data. These algorithms are "unsupervised" because they discover hidden data patterns without assistance.

The three primary purposes, for which these models are used, are to reduce dimensionality, for the association, and/or for clustering.
With clustering, unlabeled data can be sorted according to their similarities or differences using the data mining methodology. A clustering algorithm example is the K-means, Figure 3.15, which divides similar data into k groups. This is useful for image compression and data set segmentation, among other things.

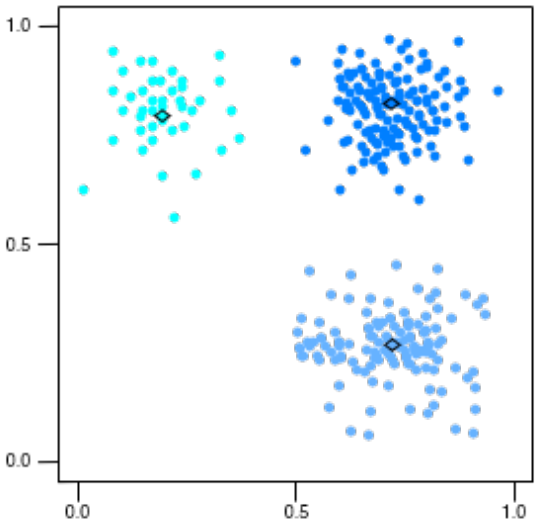


Figure 3.15: It illustrates K-means with k=3, which creates three distinct groups, each with its own centroid.

In conclusion, with Supervised learning, the model predicts data and modifies its response iteratively in order to "learn" from the training set. The unsupervised learning model operates independently to determine the underlying structure of unlabeled data. Supervised learning models are more accurate than unsupervised learning models.

Underfitting and Overfitting

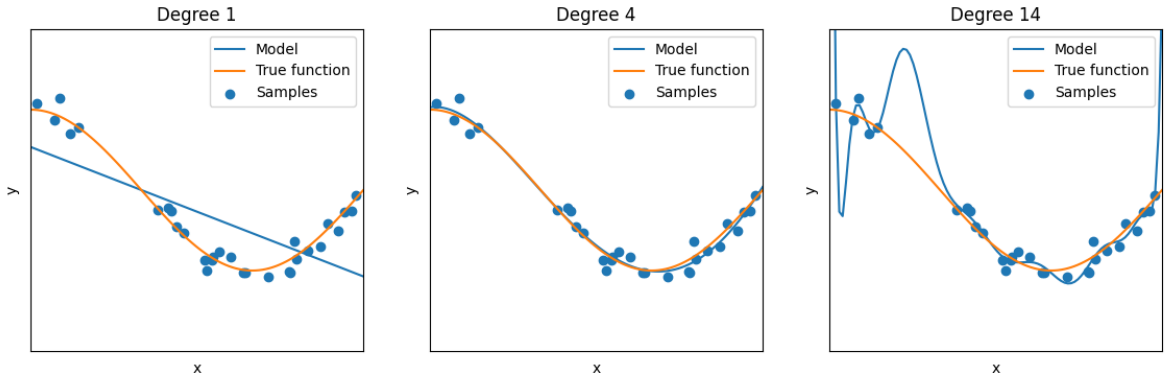


Figure 3.16: It describes Underfit and Overfit cases. The data points are represented by blue dots, the model fits by the blue line, and the real function by the orange line.

Figure 3.16 describes the underfitting and overfitting problem. The left is underfit because the model created is too simple, instead, the right is the overfit because the model does not generalize.
Underfitting can occur when the model is too simple or when the features are insufficiently helpful. In Figure 3.16, the Underfitting is illustrated by the left panel, in which the model, described by the blue line, is too simple and not generalized.

Overfitting happens when the model, during the training, is based on too many parameters and the models fits only to that specific training set, then it can not generalize

and adapts to the observed data. In Figure 3.16, it is represented by the right panel, where the model focuses too much on training dataset data and does not generalise.

Decision Tree

Decision Trees (DTs) are a model of supervised learning used for classification and regression.
The objective is to develop a model capable of predicting the value of a target variable using simple decision rules learned from the features of the dataset [47].

A decision tree is a classifier that splits the instance space recursively.
The decision tree is composed of nodes that create a rooted tree, which is a directed tree with a root node that has no incoming edges, instead, every other node has a single incoming edge. A node with outgoing edges is referred to as an internal or test node. All further nodes are known as leaves.
Each internal node in a decision tree divides the instance space into two or more sub-set based on the input attribute values. Each leaf allocates the class that represents the optimal target value. Instances are categorised by travelling them from the tree's root to its leaves based on the results of tests conducted along the journey [48].

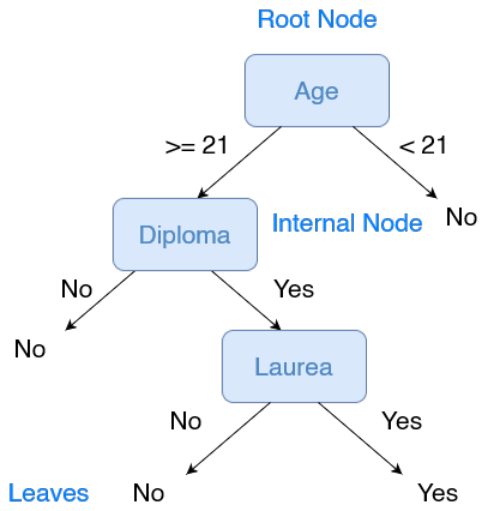


Figure 3.17: It describes the decision tree created on a dataset to classify if a person could be enrolled in the engineering master's program at the Polytechnic of Turin.

Figure 3.17 describes a decision tree used to determine if someone can be enrolled in the engineering master's program at the Polytechnic of Turin.
The decision tree starts from the Root node, then the Internal nodes are described by rectangles, while leaves are represented by the final answers (Yes or No). Each node has attributes whose corresponding values are labelled on its branches.

This model is accurate for simple datasets and the model is interpretable for small trees. It is also a fast classification and it can be scalable in training set features.
On the other hand, the decision tree can create complex trees that do not generalize the data and it can go into overfitting. To avoid that, pruning mechanisms are used such as setting the minimum number of samples at leaf note or setting the maximum depth of the tree.

The scikit-learn library [45] is used to build the decision tree with the function DecisionTreeClassifier[5].

Random Forest

The Random Forest is a bagging ensemble model, composed of several independently Decision Trees [47]; each tree makes a class prediction, and the class with the most votes becomes the model's prediction [49].
The accuracy of the result improves as the number of trees increases.

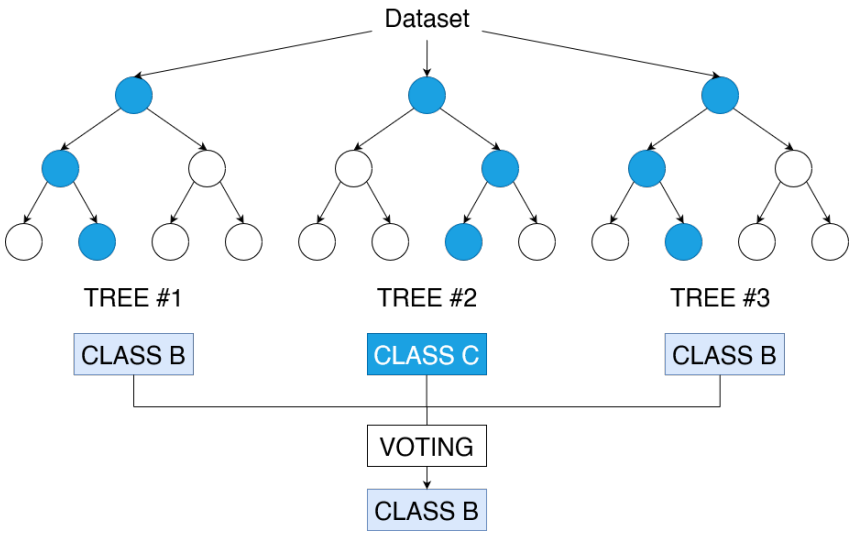


Figure 3.18: Random Forest: The dataset is split and, for each sub-dataset, is created a decision tree (tree #1, tree #2, tree #3, ...), each tree returns a class, then this class s voted by majority.

As described in Figure 3.19, the Random Forest consists of 4 steps:

1. Randomly select samples from the test dataset.
2. For each sample a decision tree is created.
3. For each tree a result is produced.
4. The result of each tree is voted by the majority.

With the Random Forest, there is better accuracy and precision, with lower overfitting of the dataset.

Random Forest is an accurate and robust method that not overfitting. It also handled missing values appropriately. It can be used to obtain the most important features in a dataset for higher accuracy.
On the other hand, Random Forest is not interpretable, because of the number of Decision Trees that are created, and with large datasets, training and prediction could be slow.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

eXtreme Gradient Boosting

Extreme gradient boosting is a supervised scalable tree-boosting system that combines the predictions of several weaker models to build a stronger model.
This method utilizes the process of "boosting", where a weak model is combined with several other weak models to produce a more robust, accurate, and stable model [50].

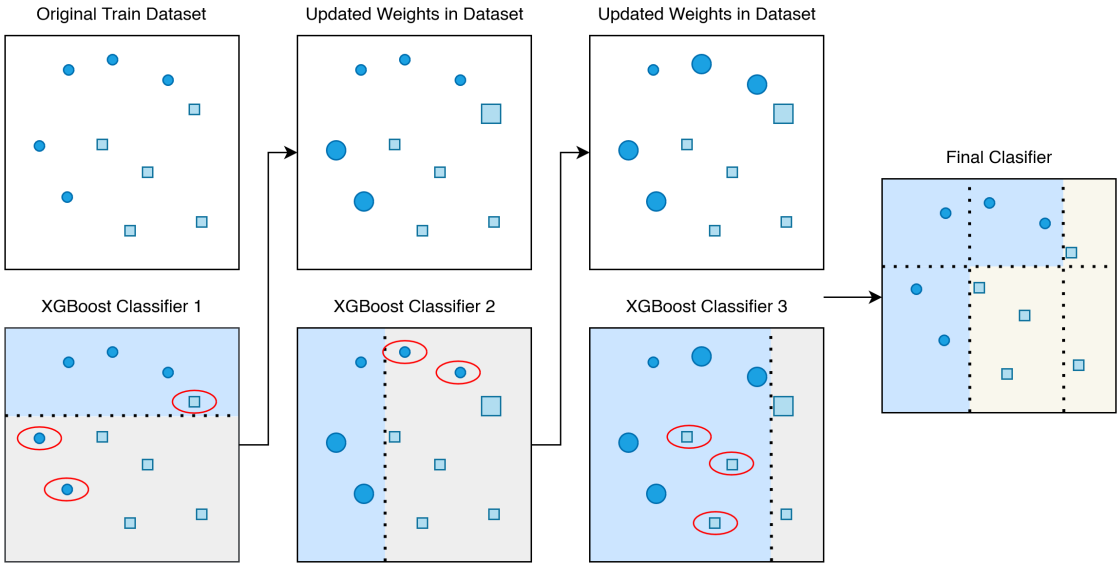


Figure 3.19: Represents the XGB algorithm Step. For each classifier, the weight of the incorrectly predicted data is increased, and the "new dataset" is then passed to a different classifier, which performs another classification and obtains a new dataset.

As described in [51], the XGB algorithm work as follow:

1. A weight is assigned to all independent variables.
2. A first weak model is created, and it generated a prediction with errors.
3. Predictions that are wrong are increased in weight.
4. The new dataset is used by the classifier to create a new model.
5. Continue until the entire training dataset is correctly predicted, when the maximum number of models is reached or when the validation loss on the dataset starts to increase (to avoid overfitting).

eXtreme Gradient Boosting is considered one of the most performed and fast algorithms. Thanks to the parallelization of tree construction which uses all CPU cores during the training, it can use also the GPU.

Based on benchmark [52], it is one of the most accurate algorithms.

3.2.4 Deep Learning (DL)

Deep Learning is a subcategory of Machine Learning that focuses on artificial neural networks modelled over the structure and functions of the human brain.

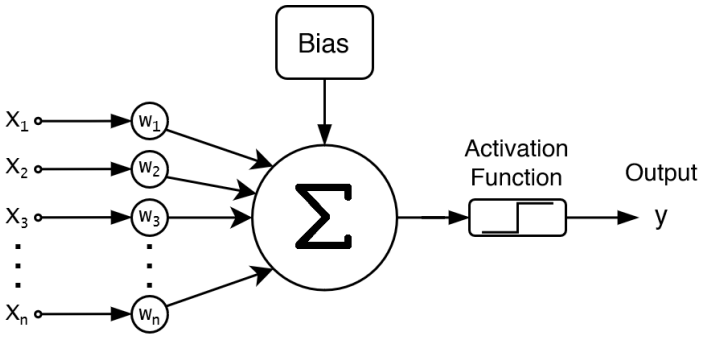


Figure 3.20: It is an Illustration of an artificial neuron. There is some input to which weights are applied, then they are summed and at the end, an activation function is applied to the output.

Figure 3.20 illustrates a neuron, the smallest component of a neural network, which is a structure that receives input values, sums them with relative weight, and then applies an activation function that simulates the biological stimulus to an input [53].

Depending on the neuron's output, the activation function determines the neuron's behaviour. Its function is to introduce a non-linearity to the output, increasing the selective capacity of the model and, consequently, the cognitive capacity of the neuron. The activation functions may help to saturate neuron outputs in fixed ranges. The main activation functions are:

- Binary Step
- Linear
- Rectified Linear Unit (ReLU)
- Sigmoid function
- Tanh function

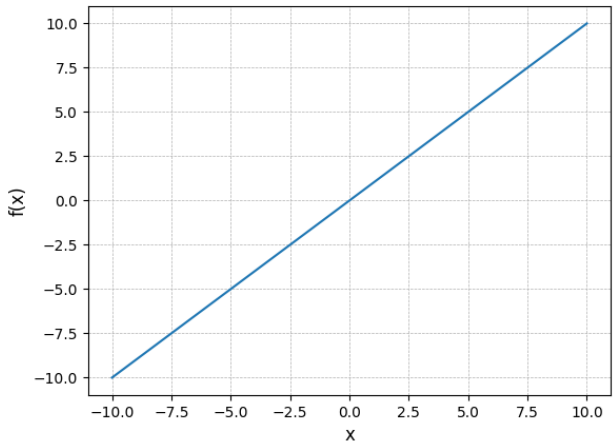


Figure 3.21: Linear function

The linear activation function is also known as the "identity function". This function does not change the input data and it returns the same value.

$$f(x) = x$$

where x is the input.

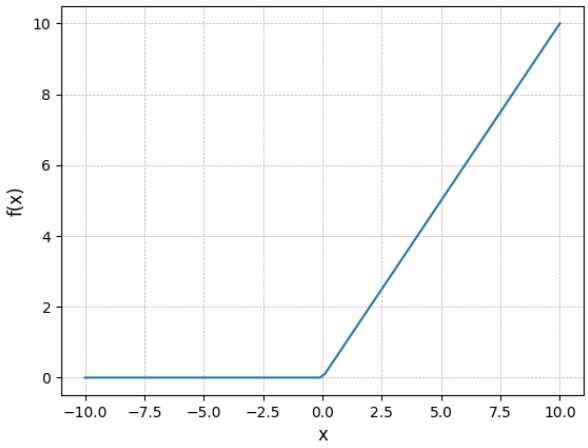


Figure 3.22: Rectified Linear Unit (ReLU) function

Rectified Linear Unit (ReLU) function, Figure 3.22, eliminates any outputs less than zero and returns the input if it is positive; if the input is zero or negative, the function returns zero and impedes backpropagation. The function is described as follows:

$$f(x) = max(0, x)$$

where x is the input.

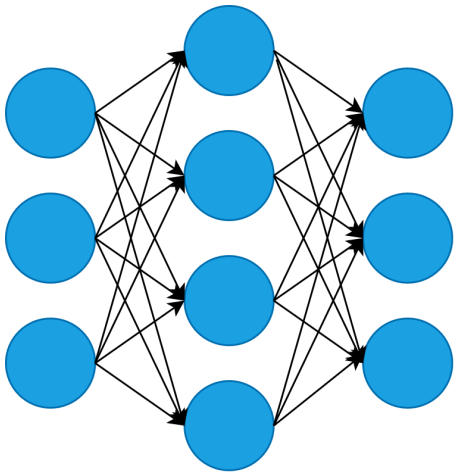


Figure 3.23: It describes a Neural Network with an Input layer, Hidden layers and an Output layer.

As illustrated in Figure 3.23, neural networks are composed of multiple layers of interconnected nodes, each of which is designed to enhance and refine prediction or classification.
External layers refer to the neural network's input and output layers, while internal layers are known as hidden layers.

The neurons in the input layer are connected to neurons in the next layer, their output is sent to the next levels after being multiplied by a weight. Each neuron, in the second layer, the first hidden layer, computes its output by summing its multiple inputs and applying an activation function and so on [53].
The final prediction or classification is made at the output level layer after the data from the input layer have been processed by the deep hidden layer of the model. The result of the output layer depends on the activation function applied to the neurons.

Many AI applications and services rely on deep learning to improve automation without human intervention.

Training

The Neural Network is trained by calculating the difference between the actual state and the desired state. Discovering the positive or negative weights of the connections between neurons is a multidimensional optimization problem in mathematics. Training the network is comparable to finding the minimum of this multidimensional "loss" or "cost" function.
The Training is performed iteratively through several training runs, altering the network's state progressively, this includes making several changes to the network's weights depending on the outputs calculated for a collection of input samples [53].

Loss Functions

A loss function is a fundamental component of the training phase, as it compares the input values to the predicted output values, determining how effectively the neural network predicts the training data and providing information on how the network learns from the training data.

Mean Squared Error (MSE) is one of the most common loss functions used for training neural network models, its formula is as follows:

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(x_i - \overline{x}_i)^2$$

where M is the number of examples in the training dataset, x is the input values and $\overline{x}$ is the predicted values.

This function is ideal for calculating loss for regression problems, as the difference is squared, regardless of whether the predicted value is above or below the target value. A disadvantage of this loss function is that it is extremely sensitive to outliers: if a predicted value is significantly larger or smaller than its target value, the loss will increase significantly [54].

Auto-Encoder (AE)

Auto-Encoder (AE) is an unsupervised Artificial Neural Network (ANN) that learns to compress data and then learn how to rebuild them, reproducing the original input. The auto-encoder has the function of ignoring the input noise and lowering the dimensions of the dataset [55].

As described in Figure 3.24, the auto-encoder consists of three parts:

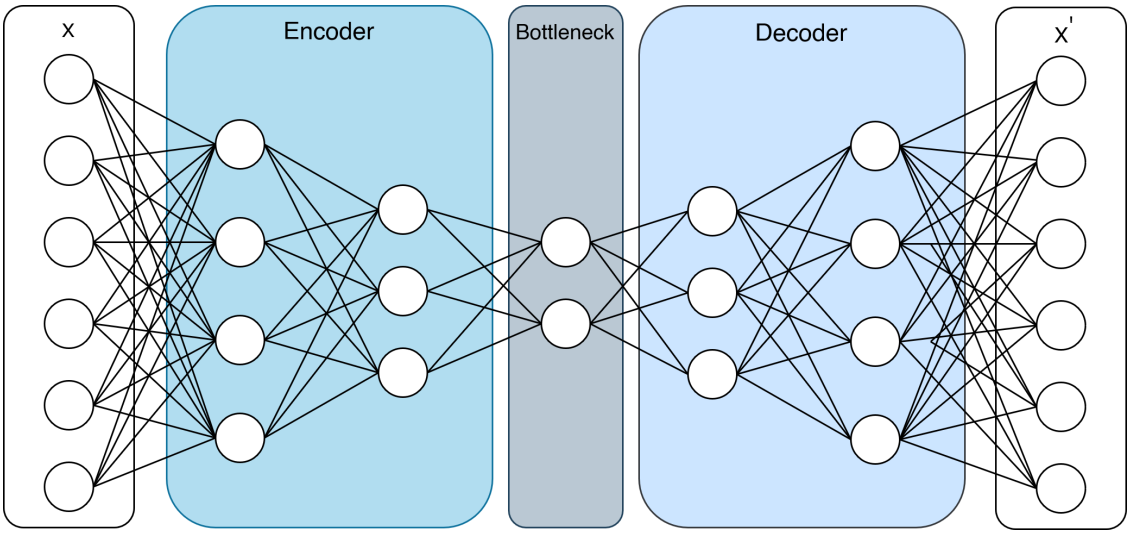


Figure 3.24: Auto-Encoder (AE), there is an Input layer, an Encoder, a Bottleneck, a Decoder and an output layer.

1. Encoder: Model learns to compress and reduce the dimensionality of input data
2. Bottleneck: is the layer with fewer neurons that you have in the auto-encoder, contains the data compressed.
3. Decoder: The model learns to reconstruct the original input data.

At the end of the process, the Reconstruction Error (RE) is used to understand how well it performs the autoencoder, and how similar the data that has been reconstructed is.

In general, the autoencoder is used to detect anomalies in a dataset. For each row, the autoencoder calculates a reconstruction error value, based on that, the anomalies have a reconstruction error which is greater than normal values. To calculate the reconstruction error values use the following formula:

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(x_i - \overline{x}_i)^2$$

where M is the number of observations in the training dataset, x is the input values and $\overline{x}$ is the value reconstructed by the decoder. The reconstruction error (RE) is the difference between the reconstructed input and its original version and is a metric that indicates how well (or how poorly) the autoencoder can recreate the input [56].

Figure 3.25 illustrates the reconstruction errors, where the x-axis is the data index, an id is the data point, and the y-axis is the reconstruction error calculated with the MSE formula explained below. This graph clearly distinguishes anomalous data (orange points) from normal data (blue points) (blue points). By entering a threshold, you can distinguish between normal and anomalous values; however, all data that falls below the threshold is considered normal, while all data that exceeds the threshold is considered anomalous.

The Autoencoder makes a dimensional reduction of the dataset, reduces the noise in the data and reduces the required space because the data are compressed. It is widely used for binary anomaly detection.

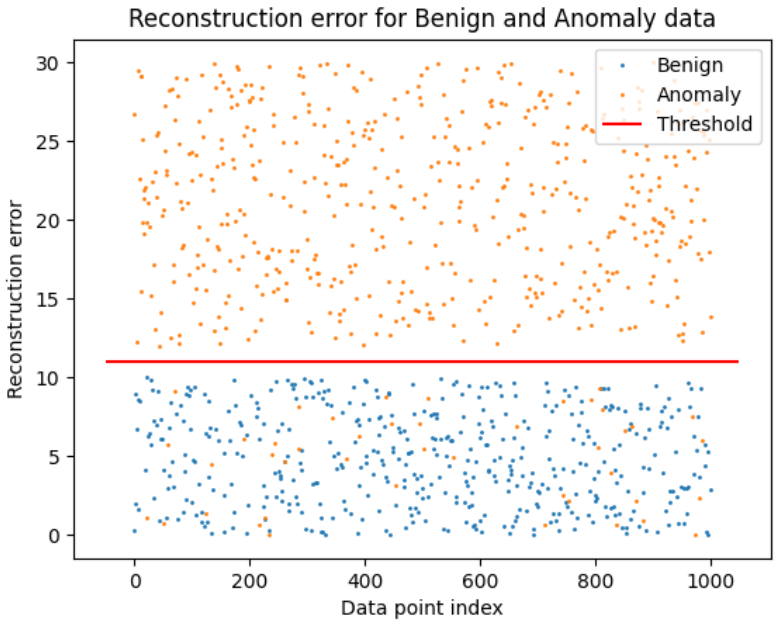


Figure 3.25: Example of reconstruction error of auto-encoder calculated with the MSE, the threshold divide the benign data from the anomaly data.

On the other hand, to do proper self-encoder training, you need to have a lot of data, depends on the layers, it can take a long time to do the training.

Chapter 4

Motivation

This chapter presents the motivations of this thesis. In particular, an Auto Encoder is selected because it can detect unknown attacks and new types of attacks and be used as a model for an IDS Anomaly-based.

The first part defines the problem, explaining the reason for creating an IDS anomaly-based model. In the second part, the state-of-the-art of the most important related works is analyzed, with a focus on their conclusions.

In the end, my contribution to this thesis is described. In particular, the proposed Auto Encoder model performs more than the one presented in the reference paper, described in the section 2.2. This model has also been tested on the CIC-IDS2017 [3], the TORSEC dataset [57], and the Heartbleed dataset to understand how it behaves if the proposed model encounters new benign traffic and if it is able to distinguish the anomalies.

4.1 Problem Definition

Starting from the Background (Section 3.1), the basic aspects of networks are analyzed, what are the possible attacks on a network and the systems that can be defended against these attacks. In particular, it is described the IDS, which monitors the network for malicious activities.

As described in the Background (Section 3.1), there are two main intrusion detection methods: signature-based and anomaly-based. The signature-based detects the attack based on their signature, so it requires an updated database also for recognizing the new attacks. Instead, the anomaly-based has the advantage that it does not depend on a database, and its database must not be updated because it is based on Machine Learning and Deep Learning techniques which know the normal behaviour of the network and are able to detect anomalies and report them.

For this reason, using an IDS anomaly-based has more advantages than an IDS signature-based, especially for new attacks for which there is still no signature.

The problem is that more models of the IDS anomaly-based are not enough exhaustive and they do not recognize the unknown attack [1]. This happens, as described in the section below because they are not trained to do that. These models are just supervised models that need to classify the traffic based on what they learned from the training phase, but if they encounter traffic that was not in the train set, an unknown attack,

their accuracy drops [58].
For this reason, there is a need for a model that can find unknown attacks in the network traffic.

4.2 State-of-Art Analysis

Established the goal to create a new model for an IDS anomaly-based, able to detect not only known attacks but also new types of attacks, the work done by several researchers is analyzed.

In the first paper, proposed in the Chapter Related Works 2.1, Canadian researchers have shown how an ensemble model composed of different Decision Tree is more performing in identifying an atypical attack (attacks with different profiles), than a single model like a Dense Neural Network (DNN) or a Linear Support Vector Classifier (L-SVC). These models use only supervised learning techniques.
Basically, it has shown that an ensemble model is able to recognize a "variant" of the attack with which it has trained. They used an attack tool present in the train-set and with which the model was trained, changed some parameters and generated an atypical attack. They do not simulate how the model behaves if there is an unknown attack, which means an attack for which the model was not trained.
Researchers define that the motivation behind their work is the absence of exhaustive IDS models against atypical and unknown attacks.

The paper described in 2.2, focuses more on detecting network anomalies. More specifically, they build an Auto-Encoder and a Variant Auto-Encoder to detect unknown attacks on the network.
The researchers do not train the model on the attacks, but only on benign traffic. This means that the model can only recognize only the benign traffic, and anything that does not recognize as benign is considered an anomaly.
The autoencoder, as described in Chapter 3.2, aims to compress data and learn how to rebuild it. Doing so generates an error, said reconstruction error. So, if the autoencoder was only trained on benign traffic, then when it sees benign traffic it knows it, making a very small reconstruction error, but if it encounters traffic that it has not been trained to recognize, generates a large reconstruction error and over a threshold considers its anomaly. To be more specific, the Auto Encoder only distinguishes normal and anomaly traffic.
In the testing process, all the classes (DoS Slowloris, DoS SlowHttpTest, Bruteforce SSH, Infiltration, Bot, ...) are re-called "anomaly", in order to understand if the methods can distinguish the unknown attacks from normal traffic. For the evaluation of the single class, they get the attack class that they want to evaluate, excluding the other attack classes, for example, to evaluate only the DoS Slowloris attack, they get all the records with the label DoS Slowloris, combined with benign traffic, and processed by the auto-encoder.

The results proposed by the paper 2.2 are pretty good. They proposed an Auto-Encoder with an overall AUC of 73% and the Variant Auto-Encoder with an overall AUC of 76%, which means that it can distinguish at 73% and 76% the Normal traffic from Anomalous Traffic.

Therefore, the Auto Encoder has the ability to detect new types of attacks and can be used for an IDS anomaly-based model.

4.3 Contribution

The proposed model in this thesis is based on the paper [2], described in Section 2.2, but more effective, in fact, as described in Chapter 8, the proposed model achieves an AUC value of 94% against the paper AUC value of 73% on the same dataset CIC-IDS2017. The same strategy to calculate the AUC values for each class is used.

The proposed model is also tested on the TORSEC dataset. In this case, is tested not only if the model can recognize the anomaly, but even to understand how the model behaves with new benign traffics, which is completely different from the dataset with which it was trained. For example, the model is trained with the CIC-IDS2017 to recognize only that benign traffic, then it is tested with the TORSEC dataset, which contains new benign traffics and anomalies traffic.

At the last the proposed model is tested on a new dataset, the Hearthbleed dataset, described in Chapter 5, using the same strategy as described above.

Chapter 5

Datasets

This Chapter explains the datasets used for the training, validation and testing of the proposed model described in Chapter 6.

In the first section, the Canadian Institute for Cybersecurity tool called CICFlowMeter[1] is discussed, which is utilized to generate the main datasets used for this thesis.

The CIC-IDS2017 dataset[2] [3], is used for the training, validation and testing phases. This dataset is used because it is one of the most complete datasets because includes the up-to-date types of attacks [59]. For this reason, it is widely used especially in the papers described in the Related Works Chapter 2. In this way, it is possible to make a good comparison with the other models proposed in the papers.

For the testing phase is also used the dataset generated by TORSEC [57] and the dataset generated with the Heartbleed vulnerability were. These two datasets are used to understand how the proposed model behaves with a different dataset for which it is not trained, and they are chosen because they contain examples that are completely different compared to the CIC-IDS2017 dataset.

5.1 CICFlowMeter

CICFlowMeter is a network traffic analysis tool created by the Canadian Institute for Cybersecurity[3]. CICFlowMeter permits the calculation of statistical values in both directions (backwards and forward); these values are generated using pcap files or in real-time by configuring the interface from which the traffic is to be obtained.

There are two modes: offline mode, used to convert a network traffic capture pcap in a csv file, or real-time mode, used to capture real-time traffic from an interface and convert it directly in a csv file.

After the CICFlowMeter has finished generating the file csv, it generates 84 features described in Appendix A.

[1] https://www.unb.ca/cic/research/applications.html

[2] https://www.unb.ca/cic/datasets/ids-2017.html

[3] https://github.com/ahlashkari/CICFlowMeter

5.2 CIC-IDS2017

Given the exponential increase in the size of networks between computers, there is a significant increase in the power of attacks on a network, as described in the introductory section, this requires new datasets to properly train Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), to counteract new recurring threats to networks.

For this need, the Canadian Institute for Cybersecurity (CIC) and the University of Brunswick (UNB) researchers decided to create a new IDS dataset called CIC-IDS2017[2] [3] that covers the main threats to network systems: DoS, DDoS, Brute Force, XSS, SQL Injection, Infiltration, Port Scan, Botnet and Heartbleed.

The dataset consists of 84 features including the label that classifies the type of traffic created. The features of the CIC-IDS2017 dataset are described in Appendix A.

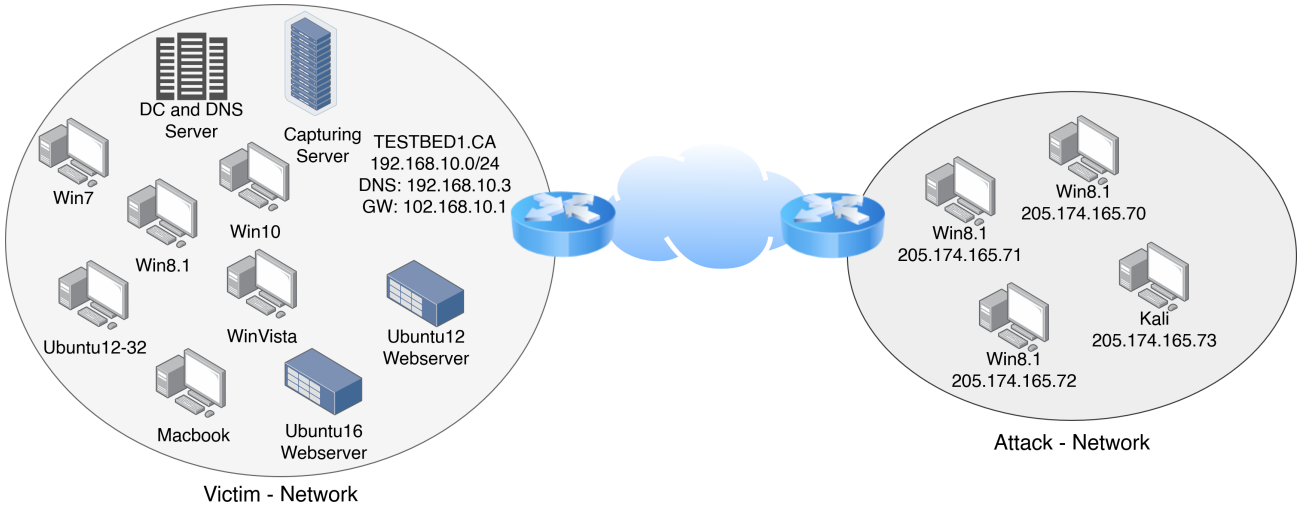


Figure 5.1: Network infrastructure created that is used as a test-bed for the creation of CIC-IDS2017 dataset. The network on the left is the "Victim network", and the one on the right is the "Attack network".

To create a working test-bed, the researchers have implemented two networks called "Attack-Network" and "Victim-Network" Figure 5.1. The Victim-Network simulates a common network that generates benign traffic, there is a Firewall, Routers, Switches and hosts with common operating systems, such as Ubuntu, Mac OS and Windows. The Attack Network is an external network, consisting of routers and switches, with a set of machines, Windows 8.1 and Kali, with Public IP addresses. In the Victim Network, there is the Capture server, on a port of the switch configured in mirror mode, the switch makes a copy of all the traffic that passes through it and sends it to the mirror port, where there is the capture server that listens to the traffic and saves it by generating pcap files. To generate constant benign traffic in the background, they used a specific profile called B-Profile, which is an agent that simulates human behaviour and it generates benign traffic.

As already mentioned, the CIC-IDS2017 dataset was created for network security and intrusion detection, for this reason, it must cover as many attacks as possible with different attack profiles. In this dataset were created six different attack profiles, which are the most common, already mentioned in Chapter 3.1 and here resumed:

- Dos Attack: The attacker tries to make an online machine unavailable, on which there is a website or an open service on a port. The machine targeted by the DoS

attack is flooded with many requests as possible, to overload the system and make the services unavailable.

- DDoS Attack: Different systems flood the bandwidth or resources of a victim machine, making the service unavailable. Usually, this attack is generated by devices infected by the attacker and that is part of a Botnet. The larger the botnet is, the more traffic is sent to the server, and the greater the attack is.
- Botnet: One or more devices infected by an attacker and become part of a Botnet. These devices are able to send data traffic generating a DDoS attack, or, because they are under the control of the malicious user, they can be used to steal sensitive data, such as passwords, from it.
- Brute Force Attack: It is a very common attack to crack passwords or can be used recursively to discover pages or hidden contents on the web.
- Heartbleed Attack: This is a Buffer Overflow attack that exploits a bug in the OpenSSL library, which is used for the Transport Layer Security (TLS) protocol. A modified request is sent to the server with the vulnerable OpenSSL Library, producing a buffer overflow and returning the data that is present in the server's memory.
- Infiltration Attack: The attacker, using a malicious file, creates a connection channel with the victim, a backdoor, in order to take control of the machine. Subsequently, it can carry out attacks (Nmap, Privilege Escalation, ...) on the internal network through the victim's machine.

The dataset is divided into several captures, which occurred at different times and several days, each of which contains a particular type of attack. The table 5.1 summarizes the days and types of simulated attacks.

Days	**Label**
Monday	Benign
Tuesday	Brute Force-FTP, Brute Force-SSH
Wednesday	DoS Slowloris, DoS SlowHttptest, DoS Hulk, DoS GoldenEye, Heartbleed
Thursday	Web Attacks, Infiltration Attacks
Friday	DDoS LOIC, Botnet, PortScan

Table 5.1: This table lists the days of traffic capture of CIC-IDS2017 with the type of Label assigned to the traffic collected on that day.

For each attack, different tools were used, in order to simulate different attack profiles. These tools are publicly available and programmed in Python.

- DoS GoldenEye: It opens numerous connections to a victim server and, with the keep-alive option, these connections are left open for long periods. With many connections, the server becomes overloaded and unable to open new ones.

- DoS Hulk: The tool sends many requests on a limited number of connections.

- Dos SlowHttptest and Slowloris: it generates attacks that belong to the category of slow DoS, ie it generates traffic on a low band, with a low transmission speed and a high transfer time for each packet. The HTTP protocol needs to receive the entire request before processing it, and the server keeps the resources occupied until the request is completed. With many incomplete HTTP requests, the server consumes the available resources to keep them active, until it overloads and becomes unavailable. Slowloris, unlike SlowHttptest, makes requests with an incomplete header.

- DDoS: The tool used to simulate this attack is Low Orbit Ion Cannon (LOIC).

- Heartbleed: To simulate this attack, they used an Ubuntu 12.04 Server with the vulnerable version of OpenSSL 1.0.1f, then the Heartleech[4] python script is used to reproduce the attack.

- Infiltration: Metasploit was used for this attack, a tool that contains the most common vulnerabilities. After the victim downloaded a malicious file, the attacker performs a port scan with Nmap on the Victim-network.

Labels	**Number of examples**
BENIGN	1652527
DoS Hulk	171974
DDoS	128011
DoS GoldenEye	10281
FTP-Patator	5931
DoS slowloris	5276
DoS Slowhttptest	5186
SSH-Patator	3153
PortScan	1922
Web Attack Brute Force	1427
Bot	1337
Web Attack XSS	652
Infiltration	36
Web Attack SQL Injection	20
Heartbleed	11

Table 5.2: This table lists all labels with the total number of examples in the CIC-IDS2017 dataset.

The traffic captured, is saved in pcap files and these are processed by the CICFlowMeter software described above, thus generating the dataset with the 84 features described in the table A.1 above.

The CIC-IDS2017 dataset has several problems [60]:

[4] https://github.com/robertdavidgraham/heartleech

1. The dataset generation process requires time and resources. The dataset is about 1GB in size, then to open with pandas[5], it requires more than 1.7GB of memory in approximately 10-15 minutes (5-7 minutes with multi-threading).

2. From the table 5.2, the first thing that can be seen is that the dataset is unbalanced, and the benign traffic is higher than the other labels. This is a problem for the training of the model, as it only provides the label with a greater number of examples, and never those with a smaller number. This leads to low model accuracy and a high number of False Positive Rates (FPR) during the validation and testing phase.

3. The dataset contains some examples with "*NaN*" or corrupt values.

5.3 TORSEC Dataset

This dataset was created by the TORSEC group of the Polytechnic of Turin, and it is not publicly available. The dataset is based on real traffic and it is divided into Benign traffic and Malicious traffic: Benign traffic is generated by Chrome, Edge and Firefox browsers, while Malicious traffic contains vulnerability scan, web crawler and DoS attacks generated with GoldenEye, Hulk, Slowloris and SlowHttptest tools [57].

The TORSEC dataset is used during the testing phase of the models, mainly are used examples of benign traffic (Chrome, Edge and Firefox), DoS attacks (DoS Slowloris, DoS Goldeneye, DoS Hulk, DoS SlowHttpTest) and Bot attack.

To construct the dataset, CICFlowMeter was used to convert pcaps to csv files; therefore, the features are the same as described in Table A.1.

Label	Number of example
Benign	20327
DoS GoldenEye	343258
DoS Hulk	193718
BENIGN	40514
DoS slowloris	5500
DoS Slowhttptest	3838
Bot	3598

Table 5.3: This table shows the type of traffic generated with the number of examples in the TORSEC dataset.

5.4 Heartbleed Dataset

This dataset was created to increase the number of Heartbleed cases in the dataset CIC-IDS2017, Section 5.2. To generate the dataset, a lab was built and the traffic generated was captured using the TCPdump[6] tool which generated three pcaps (2 benign and 1 Malicious) which were converted to .csv using the CICFlowmeter tool. The features are the same as described in A.1.

[5] https://pandas.pydata.org/

[6] https://www.tcpdump.org/manpages/tcpdump.1.html

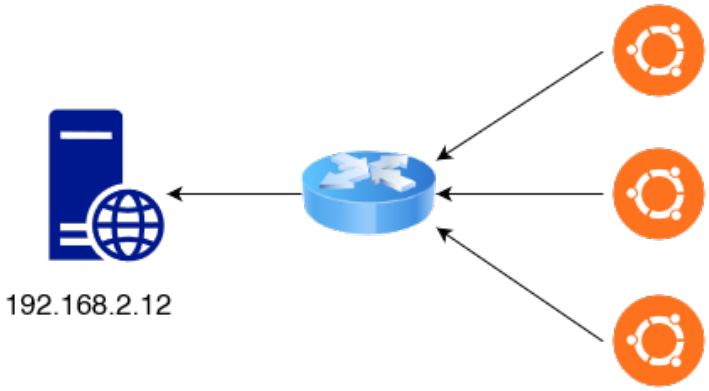


Figure 5.2: Network infrastructure created that is used as a test-bed for the benign flow.

For the benign part, a 192.168.2.0/24 network has been created, as described in Figure 5.2, in which there are 3 Ubuntu virtual machines and 1 Ubuntu virtual machine server. On the Ubuntu Server machine, there is a container docker (jas9reet/heartbleed [7]) that uses the OpenSSL version library (1.0.1) with the Heartbleed vulnerability, to open an Apache server on port 8443. The three Ubuntu virtual machines have a script written in bash that gets the page from the Ubuntu server at different times.

When you try to access to the web server, there is a login page. In some cases, the Ubuntu client has requested the login credentials. In this way was simulated real traffic with access credentials. To increase the number of benign traffic, this was combined with some parts of the benign traffic of the TORSEC dataset.

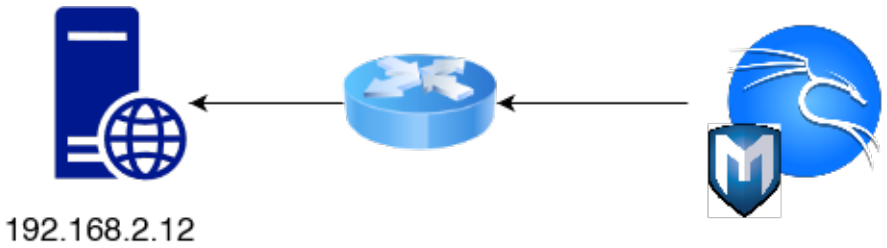


Figure 5.3: Network infrastructure created that is used as a test-bed for the Malicious flow.

For the Heartbleed vulnerability, a Kali[8] linux machine was used with Metasploit[9], then the command for exploit the heartbleed vulnerability are used[10].

```
>> msfconsole
>> use auxiliary/scanner/ssl/openssl_heartbleed
>> set VERBOSE true
>> set RHOSTS WEB_SERVER_IP
>> set RPORT WEB_SERVER_PORT_IP
>> run
```

[7] https://github.com/jas9reet/heartbleed-lab

[8] https://www.kali.org/

[9] https://www.metasploit.com/

[10] https://github.com/jas9reet/heartbleed-lab

To make all of this more automatic, a bash script was created that repeats the Metasploit commands.

Label	**Number of example**
BENIGN	2713
Heartbleed	866

Table 5.4: This table shows the type of traffic generated with the number of examples in the Heartbleed dataset.

Chapter 6

Proposed Solution

This Chapter explains how the dataset is set up, the features used after the feature selection functions, which preprocessing function I used for data and the proposed method based on Auto Encoder to find anomalies in the traffic network.

6.1 Dataset Features Selection

For the training, validation and testing phase, the CIC-IDS2017 dataset is used. It is divided into 80% of the train-set and 20% of the test-set, then the train-test is in turn divided into 80% for the training and 20% for the validation phases.

As described in the next chapter, for testing the models with the unknown attack, it is used as part of the TORSEC dataset. Then the proposed model is tested with the Heartbleed dataset.

The reason for this choice is to create a model that is as generic as possible. There is a need that the proposed model works with different datasets and is able to find the majority of the anomalies inside the dataset used which corresponds to finding the anomalies in the network, for recognizing new forms of attack.

Starting from the 84 features of the CIC-IDS2017 dataset, previously described in the 5.2, a features_selection [1] is applied, based on *RandomForestClassifier()*[2] to select the best features, thus obtaining the following Figure 6.1.

The features of Flow ID, source and destination IP addresses, protocol, and source and destination ports have been removed, as they do not give any important information to the model. In addition, for this thesis, it was considered the flow of the attacks DoS Hulk, DoS Goldeneye, DoS Slowloris, DoS SlowHttptest, SSH-Patator, FTP-Patator, Bot, Infiltration and Heartbleed from the dataset CIC-IDS2017.

For each feature of Figure 6.1, a histogram is drawn, based on the value assumed by a label and the normalized frequency that value is assumed, which means that the frequency is transformed into a range [0,1], for of the difference of examples between the labels. Based on the features histogram, if there is a distinction between benign traffic

[1] https://scikit-learn.org/stable/modules/feature_selection.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

from malicious traffic (DoS Hulk, DoS Goldeneye, Infiltration, Hearthbleed, ...), then that feature is selected, otherwise, it is discarded.
All the histograms of the selected features are in Appendix B.

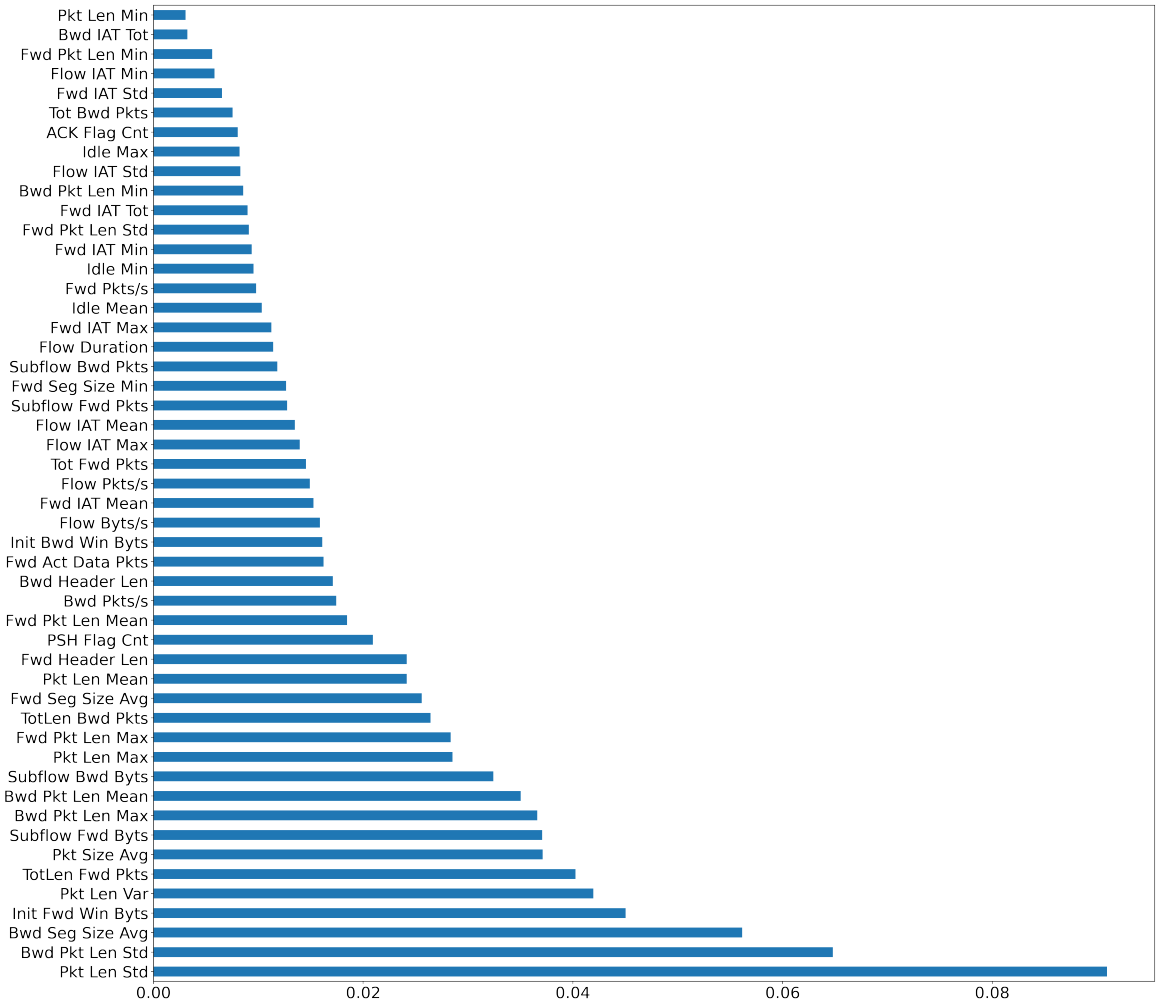


Figure 6.1: Here are described the best 50 features selected during the phase of features selected with the Random Forest. A horizontal bar plot is a useful chart for representing feature importance. The features are sorted in ascending order according to the score obtained during the feature_selection, from the least important to the most important.

After selecting the features, with feature_selection and histograms, the following Table 6.1 is obtained.

#	Name Features	#	Name Features
1	Flow Duration	11	Pkt Len Max
2	Bwd Pkt Len Max	12	Pkt Len Mean
3	Bwd Pkt Len Std	13	Pkt Len Std
4	Flow IAT Mean	14	Pkt Len Var
5	Flow IAT Std	15	Bwd IAT Tot
6	Flow IAT Max	16	Pkt Size Avg
7	Fwd IAT Tot	17	Bwd Seg Size Avg
8	Fwd IAT Mean	18	Fwd Pkt Len Std
9	Fwd IAT Std	19	Fwd Pkt Len Max
10	Fwd IAT Max	20	Bwd Pkt Len Min

Table 6.1: In this table are described all the selected features after the feature_selection and the histograms analysis.

6.2 Dataset Preprocessing

One of the problems of the CIC-IDS2017 dataset, described above in Section 5.2, is the presence of some null values. Dataset Preprocessing is needed not only to eliminate null values, which are a big problem in machine learning and deep learning but also to make the training and testing process easier for the model. For each csv file, after it is opened with the panda's dedicated functions [3], a check is made on the *Nan* values and on the data type consistency in the column, to avoid the mix of discrete values (ie, integer values) with continues values (ie, fractional value) and ordinal data (ie, strings).

After cleaning the dataset from the null values and validating each column with the respective type, the dataset is divided into two parts: x containing all the columns of the features described in Table 6.1 without the labels column, and y containing only the labels column. Next, the MinMaxScaler()[4], a function that transforms each feature to a given range, is applied to x and LabelEncoder()[5], that encode target labels, is applied to y.

As already mentioned in Section 5.2, the CIC-IDS2017 dataset is unbalanced, there is too much difference between the classes, Table 5.2, there is the need to balance it with different solutions.

There are three solutions to rebalancing a dataset:

- Under Sampling: the label with more examples (the majority classes) is removed randomly to reduce it to an acceptable number.
- Over Sampling: Examples are added to labels with fewer examples (the Minority classes) to increase to an acceptable number, the same as the majority class.
- Class Weights: Weights are calculated for each class and are taken into account during the training phase of the model.

[3] https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[5] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

The solution chosen to rebalance the dataset is the Class Weights, weights are then calculated using the following formula:

$$w_i = \frac{max(count\_all\_labels)}{count\_label_i}$$

It is the fraction ratio between the maximum number of all the labels and the number of the current label.
Subsequently, the obtained values are applied during the training phase of the model to rebalance the dataset.

6.3 Proposed Model: Autoencoder

The proposed method is an Autoencoder, described in Chapter 3.2, which uses unsupervised learning techniques.

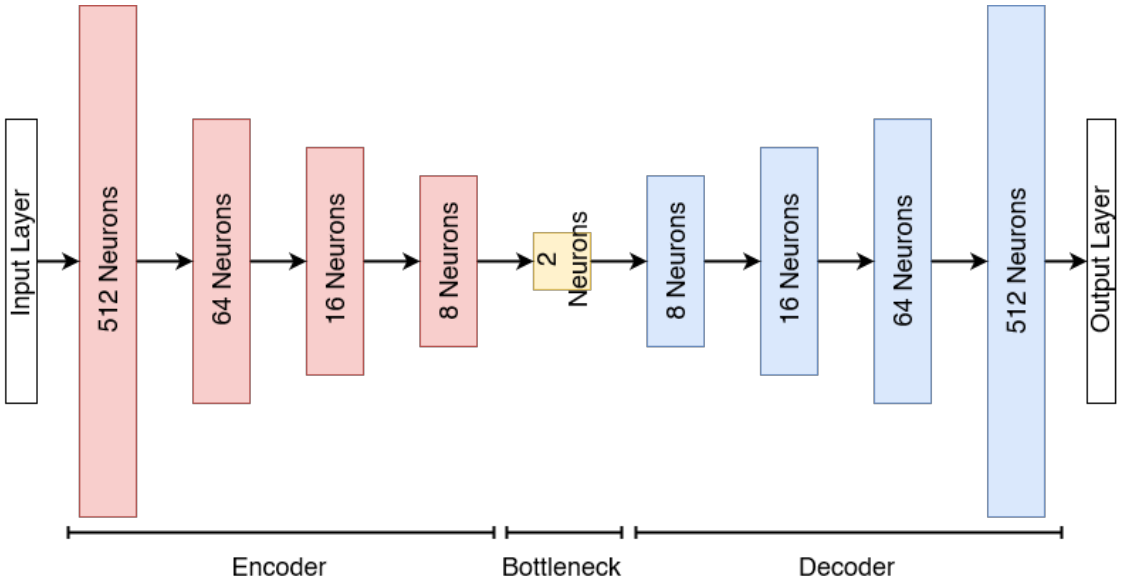


Figure 6.2: This figure describes the loss curve for the training and the validation. It is calculated with MSE.

The Auto Encoder is divided into three parts, the first part is the encoder, the second part is the bottleneck, and the third is the decode. The principle is that the encoding part compresses the input data until it reaches the bottleneck, meanwhile, the decode should reconstruct the input data, which generates an error called "reconstruction error". The encoding and decoding parts are symmetric, meaning the first encoder layer has the same neurons as the last decoder layer and the number of layers is equal.

The structure of the proposed model is described in table 6.2.

To avoid overfitting, for each layer excluded the input, the output and the Bottleneck layer, there is a Dropout layer which drops out the nodes. In this case, the dropout is set to 0.2, which means that for each layer 20% of neurons are dropped out before passing data to the next layer.

The training and validation phase is performed only on the Benign traffic, this means that all the attack labels are not considered in this step. This is because the proposed model should recognize only the Benign traffic, then, when it receives the anomaly traffic

Layer	Description
Input	[19]
Encoder	[512, 64, 16, 8]
Bottleneck	[2]
Decoder	[8, 16, 64, 512]
Output	[19]

Table 6.2: In this table is described the layer of the Auto Encoder.

it will not be recognized as benign because the model does not ever meet it during the training phase. The principle is: "The anomalous traffic is all that traffic which is not benign".

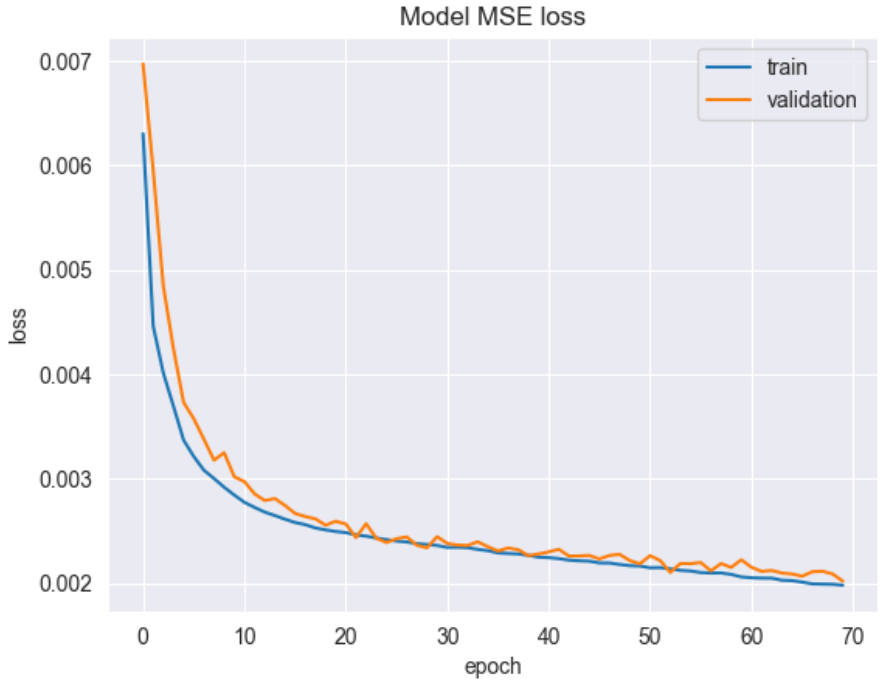


Figure 6.3: This figure describes the loss curve for the training and the validation. It is calculated with MSE.

After the training phase, Figure 6.3 is obtained which describes the loss graph. The loss curve shows what the model is trying to understand from the data, then the validation curve indicates if the model is overfitting or not and if it is learning something from the data. In this case, the validation curve follows the training curve, which is good, because the model is learning how to reconstruct the data.
The loss curve is calculated with MSE, as described in Chapter 3.2.

The dimensions of the layers and the hyperparameters were determined through trial and error. In this case, it maintained constant a parameter, such as the network layers and neurons, and tried to change the learning rate or the batch size to see if there is an improvement.
The neural network configuration parameters for the proposed method AE is described below:

1. For the learning rate $[5e^{-3}, 1e^{-3}, 5e^{-4}, 1e^{-4}, 1e^{-5}]$ were tested and $1e^{-4}$ is chosen.

2. For the batch size $[256, 512, 1024, 2048]$ were tested and 512 is chosen.

The next step is the testing phase which is described in the next chapter and is based on the reconstruction error calculated with MSE. The strategy for testing the auto-encoder is described in the following chapter: one attack at a time is used, excluding all others, and then the metrics are calculated based on the outcome of that attack. This procedure is repeated for each dataset attack.

Chapter 7

Implementation

This Chapter presents the implementation of the proposed model. The first section describes the frameworks used for developing the proposed model. The second describes the implementation with the code, the function called and their description.

7.1 Frameworks

In this section are described all the tools used to develop the proposed model.

The proposed model is created using Python 3.9.0[1] with Tensorflow 2.9.1[2] and Keras 2.9.0[3]. In particular, to reduce the training time and use the GPU, it is used the Conda version of Tensorflow 2.9.1[4] with cudatoolkit 11.2[5].

For the data preprocessing, the creation of the confusion matrix and the ROC curve, is used scikit-learn[45], matplotlib[6] and seaborn[7].

For opening the csv file it is used pandas[8] and numpy[9].

The tool CICFlowMeter[10] is used to create the csv file from the pcap files.

7.2 Methodology

This section is described how the proposed model is developed.

Figure 7.1 describes the different phases of the workflow of the proposed model:

[1] https://www.python.org/downloads/release/python-390/

[2] https://www.tensorflow.org/versions

[3] https://keras.io/

[4] https://www.tensorflow.org/install/pip#linux

[5] https://developer.nvidia.com/cuda-toolkit

[6] https://matplotlib.org/

[7] https://seaborn.pydata.org/

[8] https://pandas.pydata.org/

[9] https://numpy.org/

[10] https://www.unb.ca/cic/research/applications.html

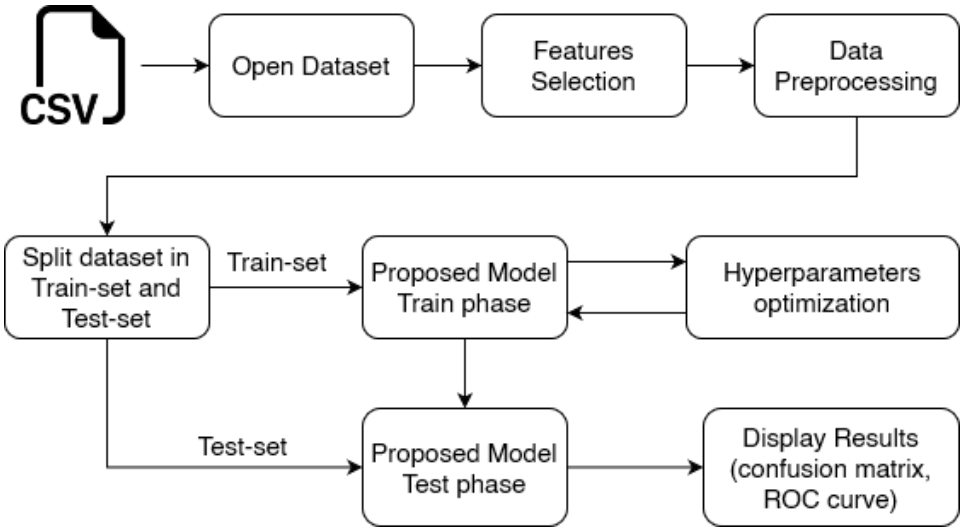


Figure 7.1: The workflow of the proposed model, from the csv of data to the display of the results.

The first step is to open the dataset; to reduce the time it takes to open the csv, a function that works in multi-threading is developed. The second step consists of selecting the most significant features, which is obtained, as previously mentioned, with the random forest classifier and the feature selection function of the scikit-learn library. Then there is the Data Preprocessing phase, in which the MinMaxScaler[11] and LabelEncoder[12] functions of scikit-learn are used to preprocess the data.

After Data Preprocessing there is the split of the dataset in train-set and test-set. To optimize the learning process of the proposed model, the test-set is used to train the model with Tensorflow and Keras functions, which also include the validation phase and Hyper-parameters optimization phase using a trial-and-error strategy. Finally, the test-set is utilized during the model testing phase.
All results are displayed at the conclusion using the matplotlib, scikit-learn, and seaborn functions.

7.2.1 Preprocessing

```
1  import pandas as pd
2  import numpy as np
3  import input_layer as iL
4
5  num_process = 8
6
7  cic17 = iL.openDatasetMultiprocessor('../../Input/cic17/Friday-*.csv',
       col_CICIDS2017, num_process)
8
9  ...
```

[11] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[12] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

```
10
11 ds = ds.append(cic17)
12 ds = ds.drop_duplicates()
```

In order to speed up the opening of the entire CSV-formatted dataset, a new function "openDatasetMultiprocessor" has been developed that uses num_process in parallel to open the dataset and reduce the time to open the dataset.
This is performed on all CSV files that have been created for each day.
At the end, the entire dataset is in the ds variable, and then the duplicate examples are dropped with the drop_duplicates from pandas.

```
1 from sklearn.preprocessing import LabelEncoder, MinMaxScaler
2
3 df_Benign = df[df['Label'] == 'BENIGN']
4 y = df_Benign.Label
5 x = df_Benign.drop('Label', axis = 1)
6
7 transformer = MinMaxScaler()
8 x_ae = transformer.fit_transform(x)
9
10 labelencoder_y = LabelEncoder()
11 y_ae = labelencoder_y.fit_transform(y)
```

This section of code there is the main part of the preprocessing of data. After it is selected only the benign traffic, using the panda's operation, it is divided into x and y. The first is the all dataset without the 'label' feature, the second has only the feature 'label'.

x is preprocessing with the MinMaxScaler[13] function by sklearn library. Each feature is scaled and translated separately by the Min and Max of the entire column.

y is preprocessing with the LabelEncoder[14] by sklearn library. It encodes the labels in a range of 0 and n_class - 1.

```
1 from sklearn.model_selection import train_test_split
2
3 x_train, x_test, y_train, y_test = train_test_split(x_ae, y_ae,
      test_size=0.2, random_state=42) # Training - Test
```

This section just divides the main dataset, composed by x_ae and y_ae, in the train-set and test-set using the train_test_split[15]. The dimension of the test-set is set with the parameter test_size, then can be also set a random_state applied to the data before the split.

7.2.2 Proposed Model

The proposed model is implemented as follows

[13] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[14] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[15] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
1  import tensorflow as tf
2  from tensorflow import keras
3  from keras.layers import Input, Dense, Dropout
4
5  ...
6
7  neurons_hidden_layers = [512, 64, 16, 8]
8
9  i = Input(shape=(number_input_layer,))
10 h = Dense(units=neurons_hidden_layers[0], activation="relu")(i)
11 h = Dropout(drop)(h)
12 h = Dense(units=neurons_hidden_layers[1], activation="relu")(h)
13 h = Dropout(drop)(h)
14 h = Dense(units=neurons_hidden_layers[2], activation="relu")(h)
15 h = Dropout(drop)(h)
16 h = Dense(units=neurons_hidden_layers[3], activation="relu")(h)
17 h = Dropout(drop)(h)
18
19 h = Dense(units=2, activation="relu")(h)
20
21 h = Dense(units=neurons_hidden_layers[3], activation="relu")(h)
22 h = Dropout(drop)(h)
23 h = Dense(units=neurons_hidden_layers[2], activation="relu")(h)
24 h = Dropout(drop)(h)
25 h = Dense(units=neurons_hidden_layers[1], activation="relu")(h)
26 h = Dropout(drop)(h)
27 h = Dense(units=neurons_hidden_layers[0], activation="relu")(h)
28 h = Dropout(drop)(h)
29 o = Dense(units=number_input_layer, activation="linear")(h)
30
31 nn = tf.keras.Model(inputs=i, outputs=o)
32
33 nn.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
       loss="mse", metrics=["accuracy"])
34 history = nn.fit(x_train, x_train, batch_size=batch, epochs=epoch,
       validation_split=0.2, shuffle=True)
```

This section describes the proposed model's main code. The input layer, which receives the shape of the number of features, is followed by the encoder part, which consists of many Dense and Dropout layers ad the length of neurons_hidden_layers array. The amount of neurons in each encoder layer is described on the neurons_hidden_layers array, and their activation function is "relu", Chapter 3.2. The bottleneck consists of only two neurons and follows the encoder. Following the bottleneck, the section is the decoder, which contains the inverse number of neurons as the encoder section. The final layer is the output, where the number of neurons is identical to that of the input layer.

Each layer receives the precedent layer as input, this means that at the end there is the input layer **i** which is composed of the only input layer, and the output layer **o** which is composed by the concatenation of all the precedent layers **h** (encoder, bottleneck and decoder) with the output layer which has "linear" as activation function.

With the Model[16] function, the "object" model in the **nn** variable can be created; however, it is the untrained model and it consists of its structure. Then, using the function Compile[17] the hyperparameters are passed. The Optimizer[18] is set, which is the algorithm for updating the model parameters such as weights and learning rate to minimise losses, as well as the function loss[19], which calculates the error that the model should reduce, and the referred metrics[20].

In the proposed model is used as optimization the Adaptive Moment Estimation (ADAM)[21] converges much faster and is far closer to the optimal than the other algorithms, such as the RMSProp[61], and as loss function MSE[22].

In the end, there is the training of the proposed model with the fit[23] function. The first element x_train is passed as input in the model, then the model generates some output from the training and that output is compared with the second element x_train, and then the MSE is calculated. Here it can be established also the epochs and the size of the validation set from the train-set.

[16] https://www.tensorflow.org/api_docs/python/tf/keras/Model

[17] https://keras.io/api/models/model_training_apis/

[18] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

[19] https://www.tensorflow.org/api_docs/python/tf/keras/losses

[20] https://www.tensorflow.org/api_docs/python/tf/keras/metrics

[21] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam

[22] https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanAbsoluteError

[23] https://keras.io/api/models/model_training_apis/

Chapter 8

Results

This Chapter presents the tests performed for the supervised model[1] (Random Forest, Extreme Gradient Boosting) and the proposed model. All the models are trained in this thesis.

Moreover, this chapter discusses how the models behave when there is a new type of attack for which they are not trained. Each model is trained with a train-set without the rows related to the new attack, for example "DoS Slowloris", and all the DoS variant attacks are renamed under the name "DoS" label, then each model is tested with the "DoS Slowloris" attack from the TORSEC dataset. This process is repeated for the other attacks (DoS GoldenEye, Dos Hulk, Bot). This phase is called "the unknown attack test".

CIC-IDS2017 has insufficient examples for Heartbleed and Infiltration, but these attacks are included for completeness and to facilitate comparison with the other models.

In the end, the Auto Encoder is tested on the Heartbleed dataset, formed by the Heartbleed vulnerability. For the test phase, the CIC-IDS2017, TORSEC and the Heartbleed dataset are used.
The main metrics used, as described in 3.2, are AUC and Balanced Accuracy, because the datasets are unbalanced [62].

8.1 Testing Phase: CIC-IDS2017

This section describes the first test performed with the CIC-IDS2017 dataset on supervised models, Random Forest and Extreme Gradient Boosting, and on the proposed model to detect attacks (DoS GoldenEye, Dos Hulk, DoS Slowhttptest, DoS slowloris, FTP-Patator, SSH-Patator, Heartbleed, Bot and Infiltration).

In this first test supervised models (RF and XGB) are trained to recognize the attacks during the test phase, that is, they have examples of attacks in both the train-set and test-set. Instead, the proposed model is trained only on recognizing benign traffic, the rest is considered an anomaly.

While the proposed model is trained to recognize only benign traffic, everything else is considered anomalous. To make comparisons with supervised methods and the reference

[1] for these models the default hyperparameters are used.

paper [2], the same approach as the reference paper [2] is used. The strategy is to test one attack at a time, excluding all others, and then repeat the procedure for all the dataset attacks. For example, only a single attack class (such as DoS Slowloris) is chosen and it is used as a test-set alongside benign traffic. After observing the results, this procedure is repeated for each class.

8.1.1 Random Forest

Here are the results obtained from the random forest during the testing phase with the dataset CIC-IDS2017.

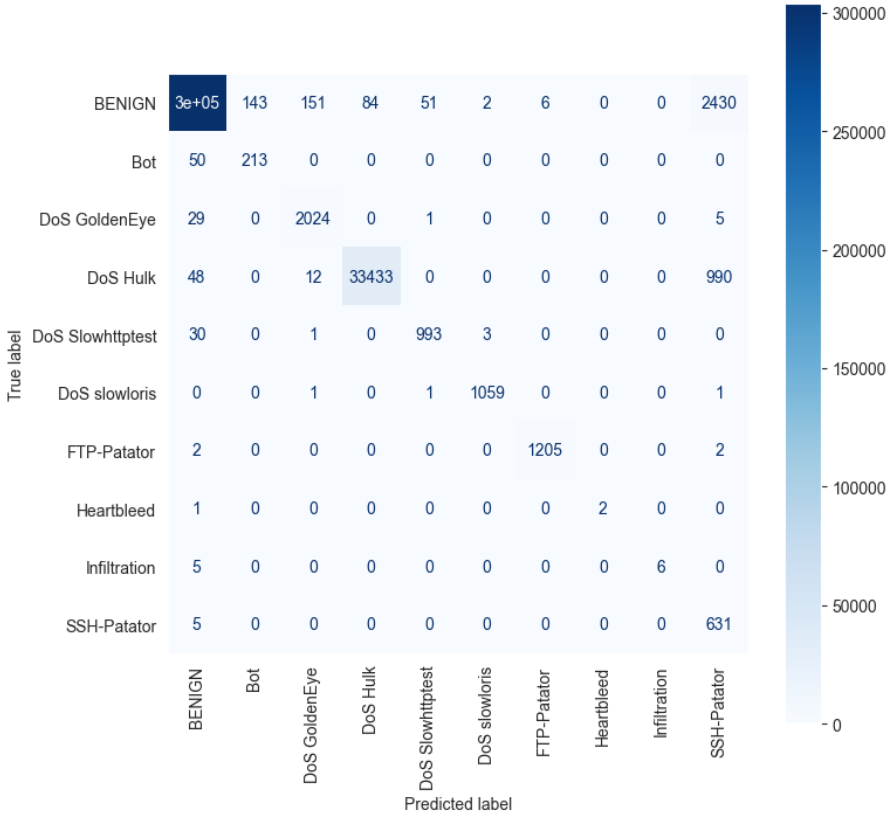


Figure 8.1: Random Forest confusion matrix with CIC-IDS2017, it shows examples that are predicted to be a particular attack class and the true labels confirms whether this prediction is correct or not.

The first thing that can be noticed from Figure 8.1 is that the dataset is unbalanced, there are too many BENIGN examples compared to the other labels. That is the reason there is a need to use different metrics instead the normal Accuracy, for Balanced Accuracy and the AUC.
The Random Forest Classifier in the testing phase, as can be seen from Figure 8.1, can predict most part of the examples proposed in the test-set. But with some labels, such as Infiltration, it generates more false positives, than true positives. However, the Random Forest Classifier has a Balanced Accuracy of 89%.

The AUC is calculated with the ROC curve for each class, as described in Figure 8.2. It is very high for each class, but for Infiltration, it is not so high, in fact, it is just 86%. This happens for the low number of Infiltration examples in the test-set, they are just 11. Because the dataset is unbalanced, it is considered the micro metrics instead of the macro

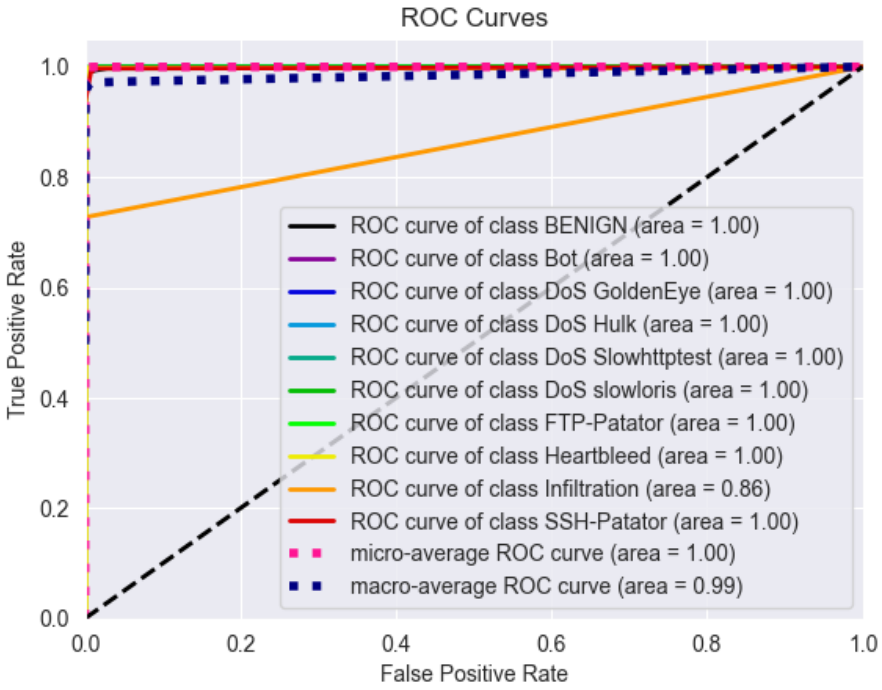


Figure 8.2: Random Forest ROC-AUC for each class with CIC-IDS2017. It shows the ability of the classifier to distinguish the positive from the negative classes.

metrics. This is because the first is weighted, instead the second is the arithmetic average, as described in Chapter 3.2.

Name Label	**AUC**
Bot	1.0
DoS GoldenEye	1.0
DoS Hulk	1.0
DoS Slowhttptest	1.0
DoS slowloris	1.0
FTP-Patator	1.0
Heartbleed	1.0
Infiltration	0.86
SSH-Patator	1.0
Micro-AUC	1.0
Balanced Accuracy	0.89

Table 8.1: Summarizes the results of Random Forest Classifier for the test with CIC-IDS2017

Table 8.1 summarizes all the results obtained for the Random Forest Classifier with the CIC-IDS2017 dataset.

8.1.2 Extreme Gradient Boosting

Here are the results obtained from Extreme Gradient Boosting during the testing phase with the dataset CIC-IDS2017. This is done to know if there are better results compared with the Random Forest.

Figure 8.3: Extreme Gradient Boosting confusion matrix with CIC-IDS2017, it shows examples that are predicted to be a particular attack class and the true labels confirms whether this prediction is correct or not.

The Extreme Gradient Boosting, as shown in the confusion matrix Figure 8.3, can predict most of the attack classes, but for someone, it has also some difficulties, for example, the DoS Slowhttptest, on 6 examples it can predict correctly only 2, the other 4 are confused with the "BENIGN" class. In general, it does not confuse the attack classes, but it confuses them with the benign class, and for that, it gets 91% of Balanced Accuracy.

About the AUC, Figure 8.4, it is high for all the classes, except for the Infiltration class. From the confusion matrix, in fact, can be seen that there are low examples for the infiltration attack, just 11 and from that it can predict only 7.

Name Label	AUC
Bot	1.0
DoS GoldenEye	1.0
DoS Hulk	1.0
DoS Slowhttptest	1.0
DoS slowloris	1.0
FTP-Patator	1.0

Heartbleed	1.0
Infiltration	0.98
SSH-Patator	1.0
Micro-AUC	1.0
Balanced Accuracy	0.91

Table 8.2: Summarizes the results of Extreme Gradient Boosting for the test with CIC-IDS2017

Table 8.2 summarizes all the results obtained for the Extreme Gradient Boosting with the CIC-IDS2017 dataset.

8.1.3 Results for Proposed Auto-Encoder Model

In this subsection, a test is performed on the model proposed in this thesis with the CIC-IDS2017 dataset.

As described in Chapter 6, the first step is to train the Auto-Encoder to recognize only the benign traffic, then, in the training and validation phase, all the anomalous traffics are discarded. In the testing phase, two strategies are used, the first one is to consider all the anomalous traffic under the same label "anomaly", and the second one, as described above, it is to consider only one attack class, excluding the others, (e.i. DoS slowloris), test the proposed model with that attack class joined with some benign traffic and then repeat the same process for the others attack classes.

Figure 8.5 shows the Reconstruction error for the anomaly test-set (the combination of all the attack classes). This is divided with the index to avoid the overlap of the points and give a better view of the graph.
This is the concept described in Chapter 6 and in Chapter 3.2. If the autoencoder is trained to recognise only benign traffic, when it sees traffic that has never been seen before, it generates a high reconstruction error. In this case, the error, for the anomaly traffic reaches 0.30, instead for the benign traffic is lower than 0.10-0.15. Then I chose a threshold with the trial and error strategy, in this case, it is 0.01, under that threshold the traffic is considered Benign, otherwise, it is considered an Anomaly.

The Figure 8.6 show the confusion matrix after the process described above. In this case, there is a good division of the true positive and true negative. It recognizes most parts of the benign traffic and it can divide it from the anomalies. The balanced accuracy is 89%.

There is a low part of benign traffic that is considered an anomaly, this happens because, as already said, for all the traffic that the auto-encoder has never seen, it is reconstructed with a high error and this happens also for the benign traffic. From Figure 8.5 can be noticed that there are some Benign points that have a high reconstructed error compared to the other point. These are anomalies because even if they are benign. For example, if the autoencoder is trained for a network that is used to generate traffic to server A and suddenly a host generates traffic to server B, the last is an anomaly, because it is outside the normal activity and it is correct to be reported even if it is benign. In this case, the network administrator can indicate the anomalies as false positives and re-train the autoencoder.

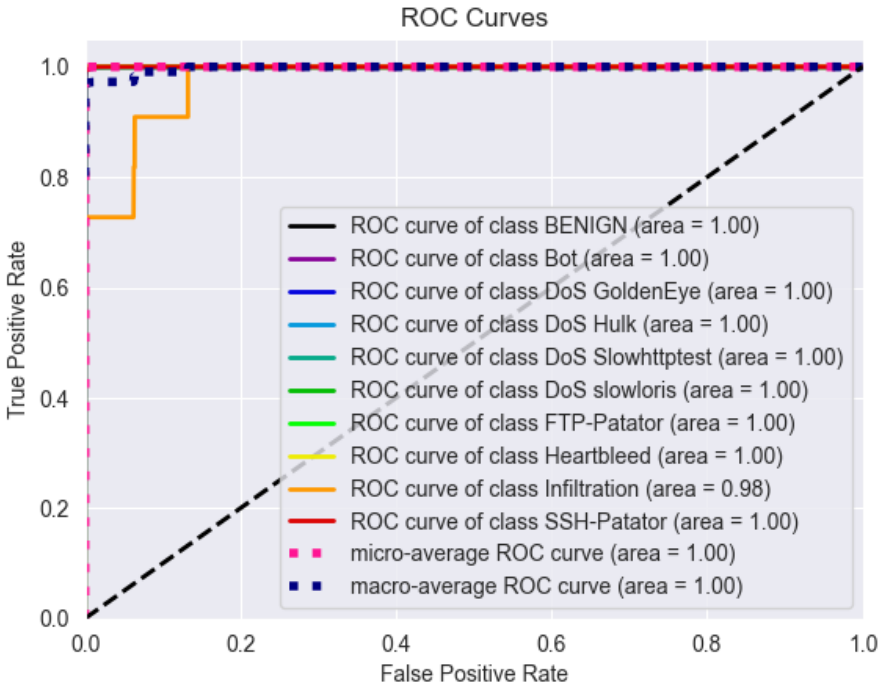


Figure 8.4: Extreme Gradient Boosting ROC-AUC for each class with CIC-IDS2017. It shows the ability of the classifier to distinguish the positive from the negative classes.

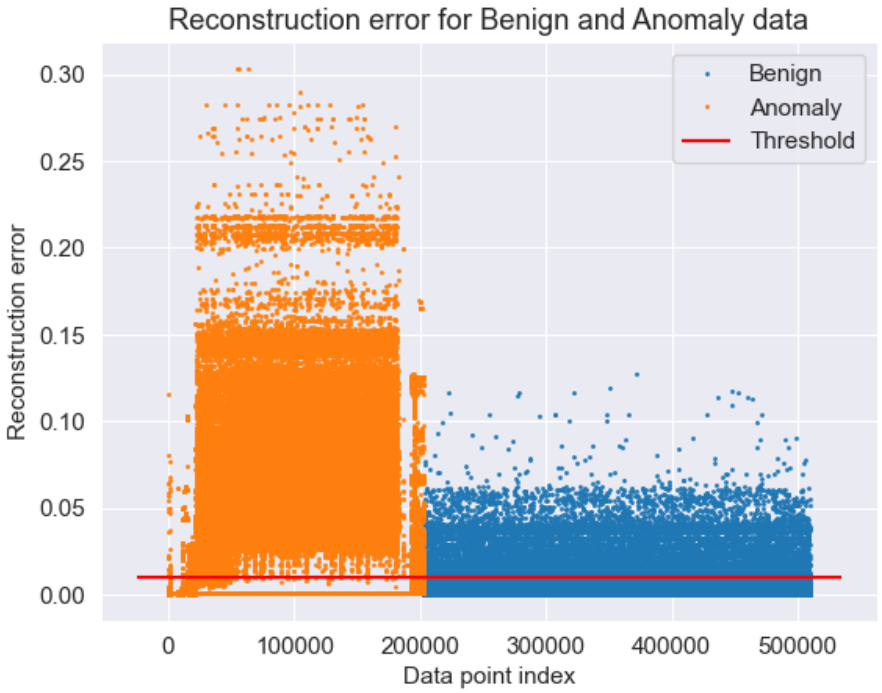


Figure 8.5: Reconstruction error for Auto-Encoder generated for the anomaly class for the test with CIC-IDS2017.

Figure 8.7 shows the AUC of the proposed model during this test phase. The proposed model obtains an AUC of 94%. In 94% of the cases, the proposed model distinguishes the positive and negative classes.

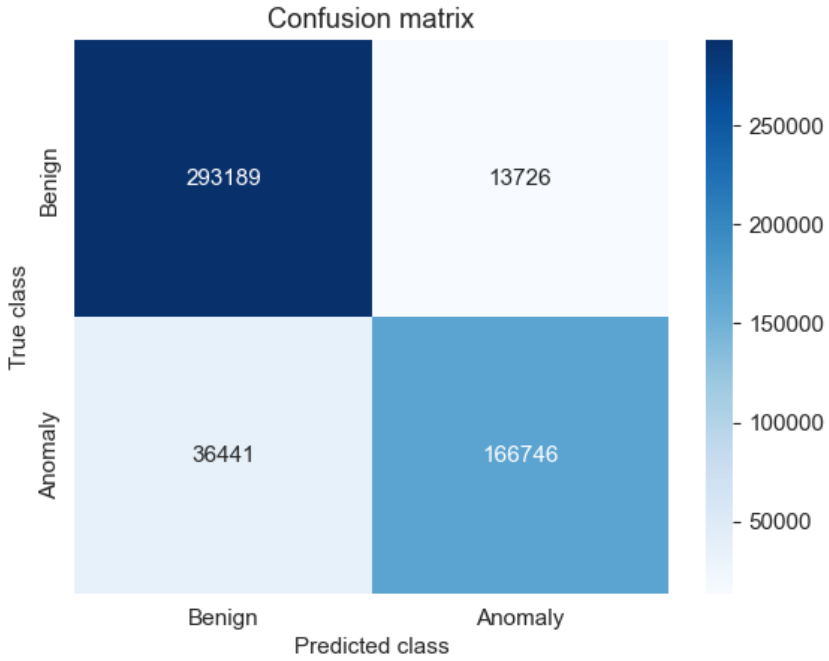


Figure 8.6: Confusion Matrix of the Proposed Model with CIC-IDS2017, it shows the true positive, the true negative, the false positive and the false negative, depending on the main class.

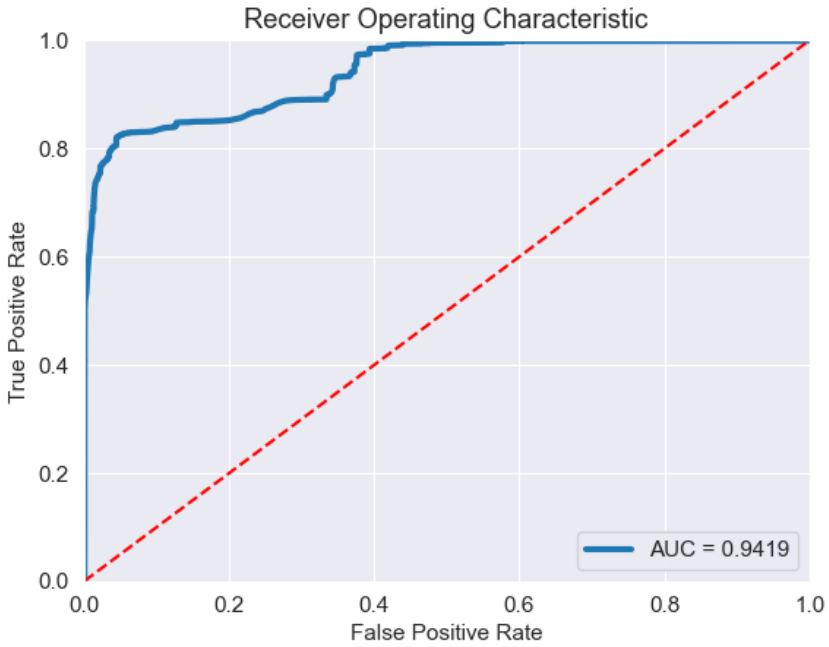


Figure 8.7: ROC-AUC of the proposed model with CIC-IDS2017, the curve is over the dotted line then it is not random.

Name Label	AUC
Bot	0.51
DoS GoldenEye	0.94
DoS Hulk	0.96

DoS Slowhttptest	0.92
DoS slowloris	0.81
FTP-Patator	0.70
Heartbleed	1.0
Infiltration	0.90
SSH-Patator	0.77
Micro AUC	0.94
Balanced Accuracy	0.89

Table 8.3: Summarizes the results of the proposed model for the test with CIC-IDS2017.

From Table 8.3, can be noticed that the model has a good value to recognize the Heartbleed attack. For this reason, this aspect is deepened below, in Section 8.3. In certain instances, such as the Bot attack, the AUC is low, as the Bot traffic closely resembles the actual traffic, and there are fewer examples compared to other attacks.

To classify the traffic, the same approach as the second reference paper [2] is used, which is to perform tests considering only one type of attack, excluding the other attaches, and then to repeat the procedure for the other attacks in the dataset. For example, select an attack, combine it with benign traffic, observe the results, and then repeat the process for the remaining classes.

8.1.4 Comparisons

In this subsection are summarized the tests performed with the CIC-IDS2017 on the Random Forest, the Extreme Gradient Boosting, the proposed model and the model proposed in the paper [2].

Name Label	**RF**	**XGB**	**Proposed Model**	**AE [2]**
Bot	**1.0**	**1.0**	0.51	0.62
DoS GoldenEye	**1.0**	**1.0**	0.94	0.75
DoS Hulk	**1.0**	**1.0**	0.96	0.83
DoS Slowhttptest	**1.0**	**1.0**	0.92	0.85
DoS slowloris	**1.0**	**1.0**	0.81	0.84
FTP-Patator	**1.0**	**1.0**	0.70	0.74
Heartbleed	**1.0**	**1.0**	**1.0**	0.98
Infiltration	0.86	**0.98**	0.90	0.89
SSH-Patator	**1.0**	**1.0**	0.77	0.68
Micro AUC	**1.0**	**1.0**	0.94	0.73
Balanced Accuracy	0.89	**0.91**	0.89	-

Table 8.4: Summarizes all the AUC results of this testing phase with the CIC-IDS2017. The best results are shown in bold.

Table 8.4 summarizes all the results obtained from this first test on CIC-IDS2017. Supervised algorithms have a higher AUC than the proposed model and the paper model [2].

The proposed model, for most classes, has a higher AUC value than the model that is present in the paper [2]. This indicates that the proposed model in this thesis is more suitable and has more potential than the model proposed in the reference paper for the CIC-IDS2017. The micro-AUC of the proposed model is 94%, instead, the micro-AUC of the model in the reference paper is just 73%. There is an improvement of 21% in the performance of the model.

The proposed model has a balanced accuracy that is higher than the XGB model and is equal to the Random Forest model. In some cases, the value of AUC between the supervised models and the proposed model is equal, for example for Heartbleed.

It is no surprise that supervised models perform better than unsupervised models. This is because supervised models are trained to classify the same labels that are in the test-set. They already learned these attack classes and they recognize them from the test-set. But, as it is explained in the paper [1] if some parameters change in the attacks, the supervised learning accuracy drops.

8.2 Testing Phase: The Unkown Attack Test

In this section, anomalous attacks will be tested.

Supervised algorithms, such as random forest and extreme gradient boosting, are trained on a train-set created with the CIC-IDS2017 dataset and then tested on a new attack from the TORSEC dataset. For example, the train-set is formed by DoS (DoS GoldenEye, DoS Hulk, DoS Slowhttptest), renamed with the "DoS" label, except for the DoS Slowloris attack that is the new type of attack. Then the test-set consists of the dataset TORSEC considering some parts of the benign traffic with the DoS Slowloris attack. This process is also done for DoS Hulk, DoS GoldenEye and Bot attacks.

The proposed model, as already described above, is trained only on the benign traffic of the dataset CIC-IDS2017 and then subsequently tested with the anomalies (DoS Slowloris, DoS Hulk, DoS GoldenEye and Bot) and benign traffic from the TORSEC dataset. After that, the proposed model is trained with the benign traffic from the TORSEC dataset and a comparison is made with the results of the proposed model trained with the benign traffic from the CIC-IDS2017. This test demonstrates that the proposed model has less performance if it is used for a different network respect to which it was trained.

As evidenced by the results, supervised models fail to recognize new attacks. Instead, the proposed model has higher performance than the supervised models which can detect new attacks.

8.2.1 Random Forest

This subsection presents the result obtained by the Random Forest for the unknown attack test. Here are reported and discussed all the graphs for the DoS Slowloris, because it is a particular case, attack, the same process is also repeated for DoS Hulk, DoS GoldenEye and Bot attacks, creating a Random Forest model for each attack.

Figure 8.8 describes the performance of Random Forest. The Random Forest can not classify the unknown DoS Slowloris attack and considers it as Benign traffic. In Fact, it predicts 2496 times in the wrong way, considering it as benign traffic instead DoS traffic. This causes a decline in performance and its Balanced Accuracy is just 48%.

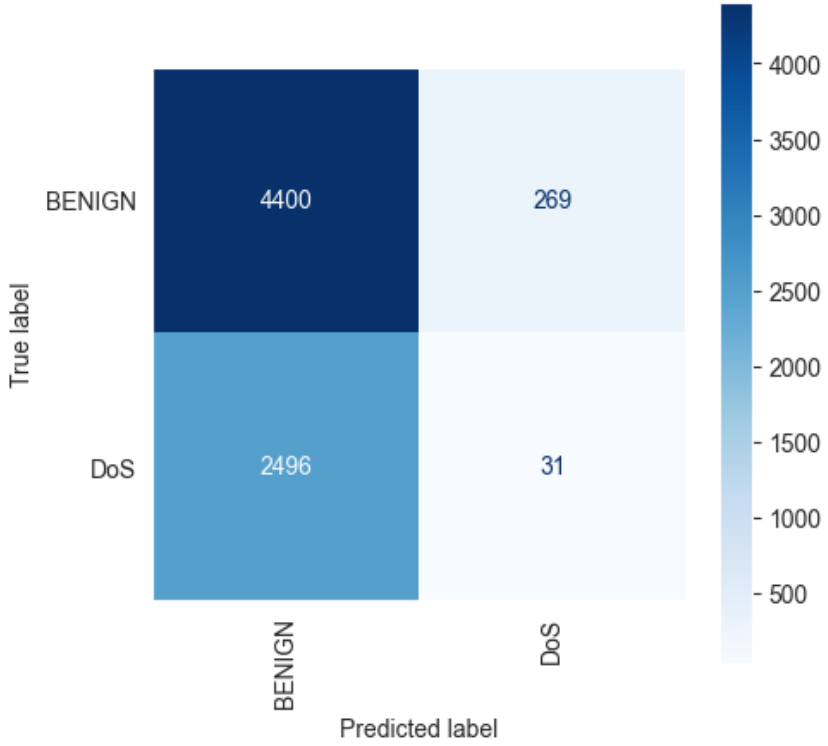


Figure 8.8: Random Forest confusion matrix for the unknown attack test of DoS Slowloris, this shows the bad performance of the classifier.

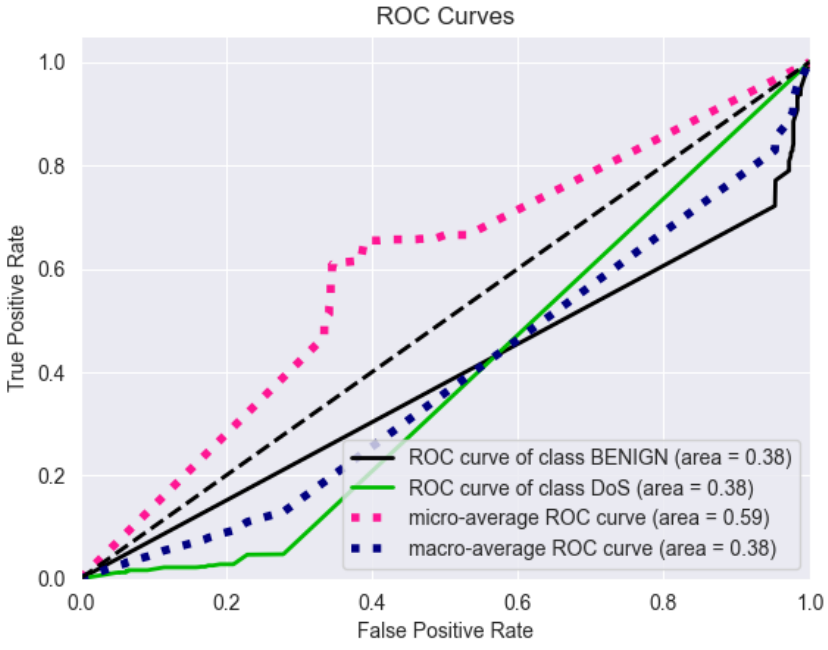


Figure 8.9: Random Forest ROC curve for the unknown attack test, this shows that the classifier is random because the micro and the macro curve is closer to the dotted.

This drop in performance, in identifying a new type of abnormal attack, can be also seen in Figure 8.9. The value for the unknown DoS Slowloris attack, the micro and macro value is closed and below the dotted diagonal. This means that the classifier acts

randomly in classifying the unknown attack, which means it can not distinguish it from benign traffic.

Name Label	micro AUC
DoS slowloris	0.59
DoS Hulk	0.19
DoS GoldenEye	0.23
Bot	0.57
Average micro AUC	0.4
Average Balanced Accuracy	0.49

Table 8.5: Summarize the results of the Random Forest for the unknown attack test.

Table 8.5 summarizes all the micro AUC obtained from the Random Forest for each attack. In some cases there are high values for the micro AUC, this is done because the model guesses correctly some labels, but all the models are completely random.

8.2.2 Extreme Gradient Boosting

In this subsection, Extreme Gradient Boosting results are described for the unknown attack test. Here are reported and discussed all the graphs for the DoS Slowloris attack, because it is a particular case, the same process is also repeated for DoS Hulk, DoS GoldenEye and Bot attacks, creating an Extreme Gradient Boosting model for each attack.

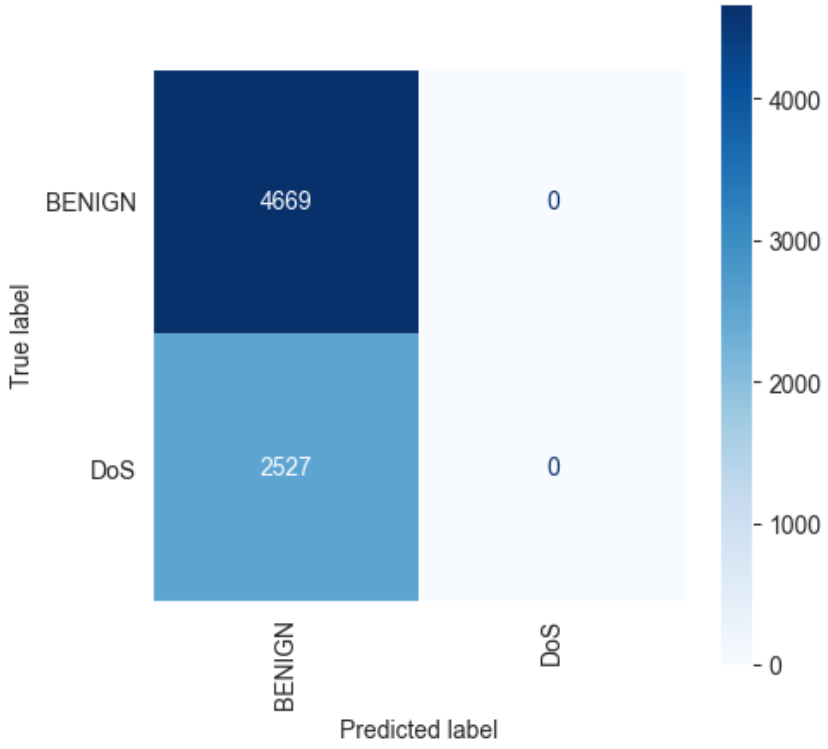


Figure 8.10: Extreme Gradient Boosting confusion matrix for the unknown attack test, this shows the bad performance of the classifier, all the traffic is predicted as Benign.

Figure 8.10 shows the poor performance of the Extreme Gradient Boosting. In this case, all the traffic is considered wrongly Benign, and in this case, it does not generate false negatives, but only false positives. From the confusion matrix considering Benign as the main class, it has Sensitivity = 1, because the False negative is 0, and Specificity = 0 because the true negative is 0, then the Balanced accuracy is 50%.

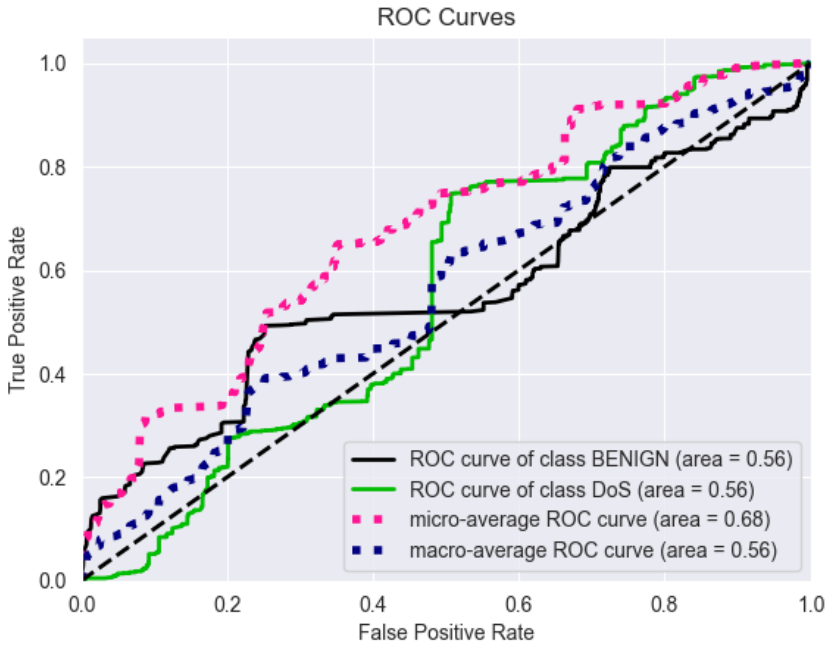


Figure 8.11: Extreme Gradient Boosting ROC curve for the unknown attack test, this shows that the classifier acts random.

The poor performance of the Extreme Gradient Boosting to find the unknown attack is also described in Figure 8.11. The macro curve, the micro curve and the DoS curve are really near to the dotted diagonal, but it is not under it, in fact, they achieve an AUC value of 0.56. For the same principle described above, this happens because the false negative and true negatives are 0. It does not classify correctly, considering all the traffic as Benign.

Name Label	**micro AUC**
DoS slowloris	0.68
DoS Hulk	0.19
DoS GoldenEye	0.30
Bot	0.59
Average micro AUC	0.28
Average Balanced Accuracy	0.50

Table 8.6: Summarize the results of the Extreme Gradient Boosting for the unknown attack test.

Table 8.6 summarizes all the results obtained by the Extreme Gradient Boosting for each attack. From the first look at the AUC, for example, the DoS Slowloris attack, it seems that the Extreme Gradient Boosting has good performance, but it is not so. It does not guess the DoS Slowloris and the other attacks, it just predicts all the traffic as

Benign, and it is a random model. This can be also seen from the Average Balanced Accuracy which is 0.5.

8.2.3 Results for Proposed Auto-Encoder Model

In this subsection, the proposed model results are described for the unknown attack test. Here is shown and discussed only the graph relative to the DoS Slowloris attack, because it is a particular case, but the process is also repeated for DoS Hulk, DoS GoldenEye and Bot attacks.

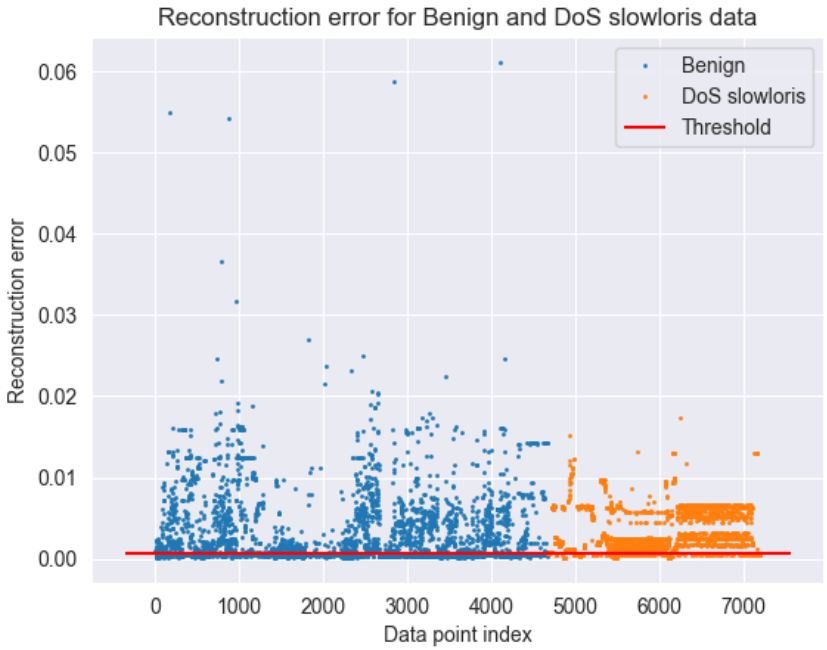


Figure 8.12: Reconstruction error for Auto-Encoder generated for the unknown DoS Slowloris attack.

Figure 8.12 shows the reconstruction error for the unknown test with the TORSEC dataset. The first thing that can be noticed is that the majority of the unknown DoS Slowloris attack is over the Threshold and then it can be considered an Anomaly. It can be noticed also that some Benign traffic is also over the Threshold and sometimes it has a construction error that is higher than the DoS Slowloris Anomaly. This happen because, as already said in the precedent test, the benign traffic of the test-set is different from the benign traffic of the train-set, so the auto-encoder looks at this traffic for the first time, it has never seen before during the train test and it does not recognize it, for that reason it generates a such high error.

This test is another confirmation that the auto-encoder should be trained on the usual traffic of the network, otherwise it considers as anomalous which is not completely wrong.

Figure 8.13 shows that the proposed model, even if it was not trained on the benign traffic of the TORSEC dataset, can recognize and distinguish the benign traffic from the DoS Slowloris, moreover it can find most parts of the unknown DoS Slowloris attack. It has a Balanced Accuracy of 68%.
This result depends also on the threshold, which is calculated with the trial-and-error strategy. In a real scenario, after the auto-encoder is trained with the usual traffic, then the threshold is established and it remains fixed because the dataset does not change. In

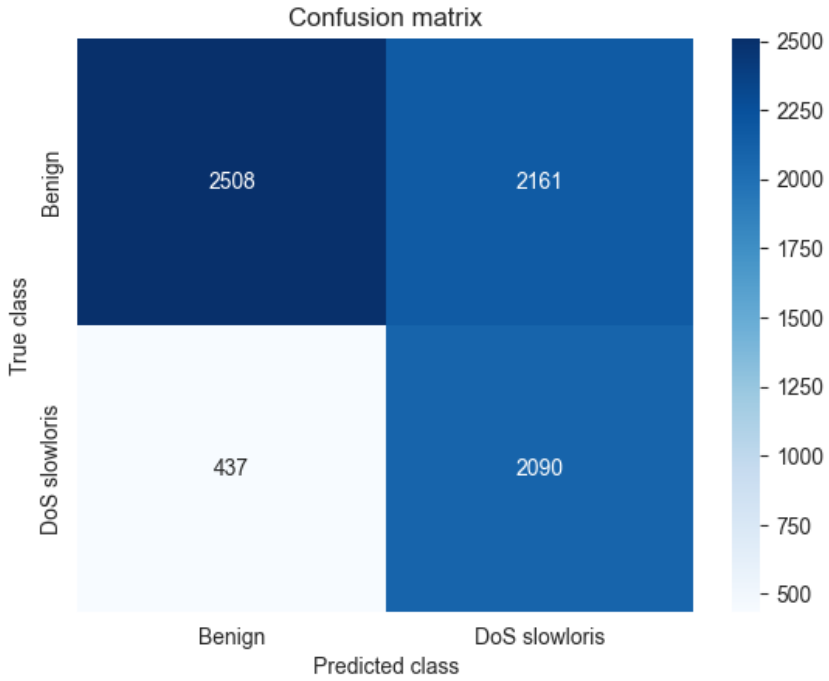


Figure 8.13: Confusion Matrix for the Proposed Model for the unknown attack test.

this test, the test-set has a different origin dataset respect to the train-set (the first one is the TORSEC dataset, and the second one is the CIC-IDS2017), so the threshold should be revised to adapt it to the scenario.

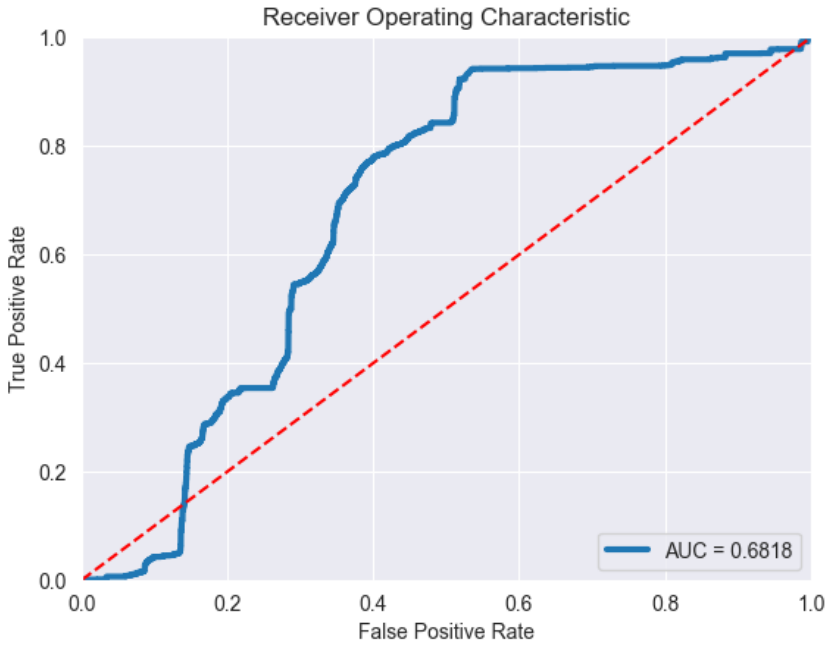


Figure 8.14: The ROC Curve of the proposed model for the unknown attack test.

The performance of the proposed model is also shown in Figure 8.14. The ROC curve is, for the majority, over the dotted line, which means that the model is not random and it can distinguish the positive classes from the negative classes. This produces a value of AUC of 0.68.

Name Label	micro AUC
DoS slowloris	0.68
DoS Hulk	0.59
DoS GoldenEye	0.61
Bot	0.53
Average micro AUC	0.60
Average Balanced Accuracy	0.63

Table 8.7: Summarize the results of the proposed model for the unknown attack test trained with the Benign traffic from the CIC-IDS2017 dataset.

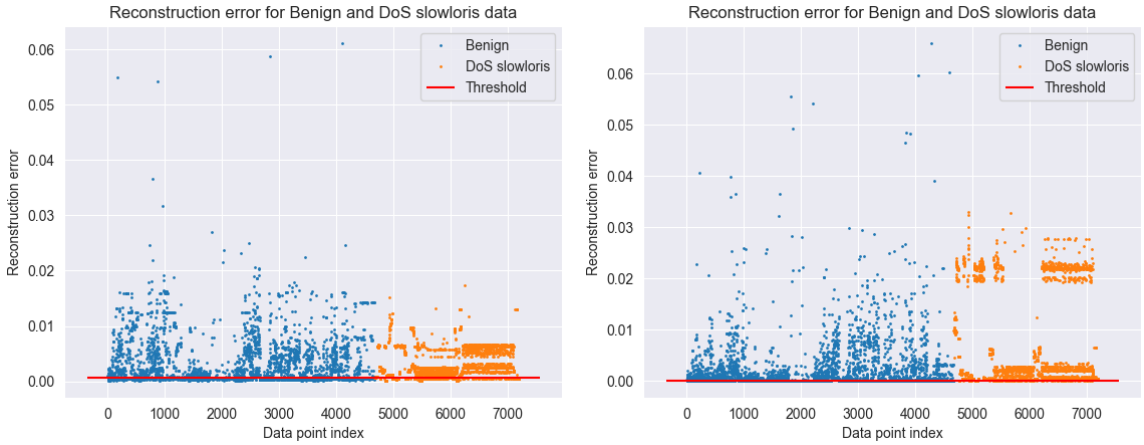
Table 8.7 describes the results obtained from the proposed model for the unknown attack test. This model, as described above, it is trained with the benign traffic from the CIC-IDS2017 and tested with the TORSEC dataset. The results are acceptable, but they are not optimal, because is the same thing to train the proposed model to recognize the benign traffic of a network and then move it to another network with new benign traffic, that the model has never seen before.
For that reason, the proposed model is trained with the Benign traffic from the TORSEC, repeats the test using the same procedure, and at the end makes the comparisons.

Name Label	micro AUC
DoS slowloris	0.72
DoS Hulk	0.69
DoS GoldenEye	0.62
Bot	0.47
Average micro AUC	0.63
Average Balanced Accuracy	0.64

Table 8.8: Summarize the results of the proposed model for the unknown attack test trained with the Benign traffic from the TORSEC dataset.

Table 8.8 summarizes the results obtained from the proposed method trained with the Benign traffic from the TORSEC dataset. There is a strange effect, all the attack performance increases, instead of DoS slowloris and Bot.

From Figure 8.15, there is a difference between the proposed model with the two datasets. The first thing is that the proposed auto-encoder model generates more reconstruction errors with the TORSEC dataset than with the CIC-IDS2017 dataset. This is because the proposed auto-encoder model, described in Chapter 6, was designed and optimized specifically for the CIC-IDS2017, and this model is too big and scattered for being trained with the TORSEC dataset. The second reason is that the CIC-IDS2017 has more examples of benign traffic than the TORSEC dataset (1652527 examples from the CIC-IDS2017 and 20327 examples from the TORSEC dataset).
On the other hand, because the train-test and the test-set are from the same dataset, meaning they are built from the same network, the proposed auto-encoder model, trained with the benign traffic from the TORSEC dataset, generates a higher reconstruction error for different attacks, and it can recognize them from the benign traffic with better performance. The main problem remains that it is necessary to build an ad-hoc model, following the proposed auto-encoder method, for the TORSEC dataset.



(a) Trained with CIC-IDS2017

(b) Trained with TORSEC

Figure 8.15: Reconstruction error for the proposed Auto-Encoder method for the unknown DoS Slowloris attack with the CIC-IDS2017 and the TORSEC datasets.

8.2.4 Comparisons

In this subsection, all the results obtained in the Unknown Attack Test for the Random Forest, the Extreme Gradient Boosting and the proposed model (trained with CIC-IDS2017 and TORSEC datasets) are summarized and analyzed.

Name Label	**RF**	**XGB**	**Proposed Model (CIC-IDS2017[2])**	**Proposed Model (TORSEC[3])**
DoS Slowloris	0.59	0.68	0.68	**0.72**
DoS Hulk	0.19	0.19	0.59	**0.69**
DoS GoldenEye	0.23	0.30	0.61	**0.62**
Bot	0.57	**0.59**	0.53	0.47
Average micro AUC	0.4	0.28	0.60	**0.63**
Average Balanced Accuracy	0.49	0.50	0.63	**0.64**

Table 8.9: Summarizes all the AUC results of the Unknown Attack testing phase. The best results are shown in bold.

Table 8.9 summarizes all the results of the Unknown Attack Test. In this test, the proposed model performs better than the Random Forest and the Extreme Gradient

[2]Trained with the CIC-IDS2017 dataset

[3]Trained with the TORSEC dataset

Boosting. This test shows how performed the proposed model (trained with the CIC-IDS2017 dataset or with the TORSEC dataset) is to find new attacks, compared to the supervised models such as Random Forest and Extreme Gradient Boosting.

This happens because the supervised model is not trained to classify these new attacks, even if they are trained with other attacks of the same categories (DoS GoldenEye, DoS Hulk, DoS Slowhttptest). In some cases, only the Random Forest classifies correctly some records, instead, the Extreme Gradient Boosting does not classify any of that records and it predicts all the records as benign. This is why both the models achieved a low Average micro AUC and a low Average Balanced Accuracy. In some cases, for example, the Extreme Gradient Boosting obtains the highest value for the micro AUC of the Bot attack and the same value for the DoS Slowloris attack. This happens because the Extreme Gradient Boosting just assigns the Benign label to all the traffic independently without logic, and for these cases, the Benign labels are many more than the attack labels. Both of these supervised models act randomly.

The proposed method recognizes the most records and for that, it has a better performance compared to the other models. It also classifies correctly the benign traffic from another dataset that it has never seen before, because, as already said, it is trained with the benign traffic from the CIC-IDS2017. In fact, if the model is trained to recognize the benign traffic from the TORSEC dataset, the performance, as shown in Table 8.9, increases because it understands the benign traffic and it is easier for the model to distinguish the anomalous traffic. This is also confirmed by the highest value of Average micro AUC and Average Balanced Accuracy. There is also to be said that the proposed auto-encoder model is optimized with the CIC-IDS2017, this is why some attacks have a similar value of micro AUC, for example, the DoS GoldenEye and Bot attacks. To achieve better results should be created an ad-hoc auto-encoder model for the TORSEC dataset.

8.3 Testing Phase: Auto Encoder and Heartbleed

This section has described the test of the proposed model on the Heartbleed dataset, described in Chapter 5. This test is done to understand how the proposed model behaves with the Heartbleed attack since with the CIC-IDS2017 Test the proposed model achieved 1.0 of AUC.

In this test, the proposed model is trained with the benign traffic from the CIC-IDS2017, and then it is tested with the Heartbleed dataset which is composed of benign traffic and the heartbleed attacks.

The proposed model performed with this dataset is described at the end.

The first thing to be considered is the reconstruction error shown in Figure 8.16. In this Figure 8.16 there is a distinct division between the Benign and the Heartbleed anomaly, just a few benign records are over the Threshold, which is also set using the trial-and-error strategy.

Figure 8.17 describes the confusion matrix of the proposed model for the testing phase with the Heartbleed dataset. In this case, there are few false negatives, considering the Benign class as the main class, and just some false positives. This means that the proposed model recognizes the benign traffic of the Heartbleed dataset, considering it has never seen that traffic before because it is trained with the benign traffic of the CIC-IDS2017 dataset, and it distinguishes the Heartbleed anomalies from the benign traffic. In this test, the proposed model has 88% of Balanced Accuracy.

Figure 8.16: The reconstruction error of the proposed model for the Heartbleed dataset.

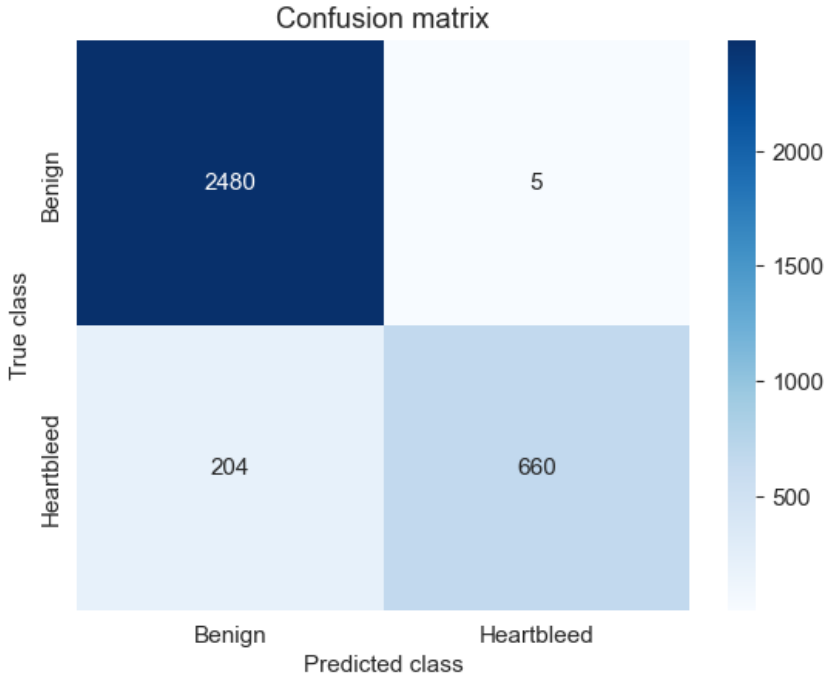


Figure 8.17: The confusion matrix of the proposed model for the Heartbleed dataset.

The ROC curve is shown in Figure 8.18. This curve is over the dotted line, so the model is able to distinguish the positive and negative classes. It achieves an AUC value of 0.85, which means that the model recognizes the Heartbleed anomalies and it distinguishes them from the benign traffic in 85% of cases.

To sum up, the proposed model is able to find the Heartbleed anomalies in the dataset

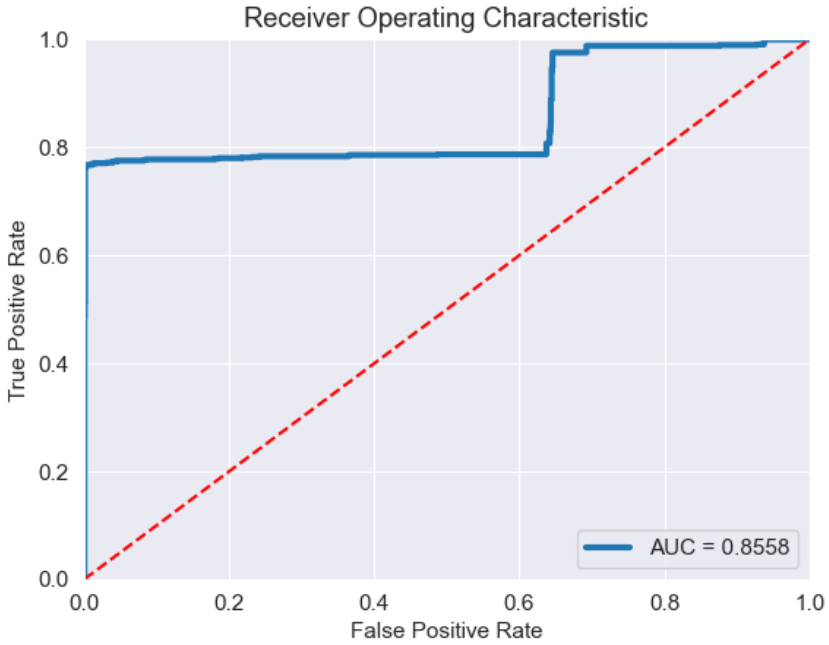


Figure 8.18: ROC Curve of the proposed model for the Heartbleed dataset, the curve is over the dotted line, then the model distinguished the positive and the negative classes.

with high performance, this means that the model can be used to find not only the Heartbleed attack but also if there is a new type of attack that is based on the same mechanism of Heartbleed. In this test the proposed model recognizes also the benign traffic even if it is not trained to do that, generating only 4 false negatives, and it distinguishes the benign traffic from the heartbleed anomalies.

Chapter 9

Conclusions

This thesis proposes a model using artificial intelligence techniques which is able to recognize anomalies and new attacks in a traffic network. In particular, it is chosen as an Auto-encoder based on unsupervised learning and can be used as a model for an IDS Anomaly-based. The proposed model is compared with two supervised models, Random Forest and Extreme Gradient Boosting. The supervised models perform better than the proposed model in finding the attacks for which they are trained, but they do not have the same performance as the new attacks that are not in the train-set. In this last case, the analysis of the proposed model results demonstrates that it performs better than the supervised model and recognizes the new attack.

As explained in Chapter 4, there is a need to identify traffic anomalies and new types of attacks due to the absence of models that can do that. These new types of attacks may go unnoticed by signature-based IDS systems because they require an up-to-date database, and if there is no signature for the new type of attack, the system can not identify it. The anomaly-based IDS has an advantage over the signature-based IDS because it does not require an updated dataset and employs artificial intelligence techniques to detect anomalies and new attack types.

On the other hand, the core problem with anomaly-based IDS is that some models are supervised, meaning they can only recognize the attack for which they were trained. If there is a DoS GoldenEye in the train-set, it can only recognize the DoS GoldenEye. However, if there is a variant of this attack or a new attack of the same category, such as the DoS Slowhttptest, the supervised modes do not detect the attack. This occurs because the model has never encountered this attack before, so it can not classify it as generating a false positive or negative.

In Chapter 8, the proposed model's performance compared to the supervised models is described. First of all, this thesis uses three datasets: the CIC-IDS2017 dataset, the TORSEC dataset and the Heartbleed dataset. For each of these datasets, a test is performed.
In the first test, only CIC-IDS2017 is used; therefore, supervised models (Random Forest and Extreme Gradient Boosting) perform better than the proposed model. In this case, there are no new attacks, and the supervised models have better performance because they are trained to recognize the examples in the test-set.
In this test, it is important to note that the proposed model is more suitable than the model proposed in the reference paper; consequently, there is an improvement in revealing know-attacks in CIC-IDS2017 using the same strategy but a different model.

The second test is titled "The Anomaly Attack" and examines how models respond to an attack for which they have not been trained. For the supervised models, only DoS and

Bot attacks from CIC-IDS2017 are used for training, excluding one attack, considered a new type of attack. Then, all the models are evaluated using a subset of the TORSEC dataset containing benign and new types of attack records. This process is repeated for each attack (DoS Slowloris, DoS Hulk, DoS GoldenEye, or Bot).
In this test, the proposed model performs better than the Random Forest and the Extreme Gradient Boosting. This is confirmation that the proposed model overcomes the limitation of these supervised models (Random Forest and Extreme Gradient Boosting), which prevents them from revealing new types of attacks.
The graph of the reconstruction error of the proposed model for this test has a particular effect: some benign records exceed the threshold and are considered anomalies. It is not entirely incorrect to do, even if it is a false positive, considering DoS slowloris the main class. The proposed model is trained to recognize benign traffic from the CIC-IDS2017 dataset and not the TORSEC dataset. This means that the proposed model is observing this traffic for the first time, producing a high error rate. If the proposed model is trained on a network that generates traffic only to a server, and one host suddenly generates traffic to a different server, the latter is an anomaly. Depending on the anomaly, the network administrator may or may not consider it an attack. If the anomaly is considered benign, then the proposed model should be retrained with the anomaly.
The second part of this test also demonstrates that the proposed auto-encoder model, trained to recognize the benign traffic from the TORSEC dataset, has better results than the proposed auto-encoder trained with the benign traffic from the CIC-IDS2017 dataset.

From the first test, the proposed model achieved the same results as the supervised models in detecting Heartbleed attacks. Therefore, a third test is performed with the dataset created ad-hoc named Heartbleed, Chapter 5. This is to understand how the proposed model behaves with these attacks.
This test demonstrates that the proposed model achieves high-performance detecting and separating Heartbleed attacks from benign traffic.

This thesis aims to identify network anomalies and new attack types using artificial intelligence techniques.
The proposed auto-encoder model, and unsupervised learning in general, is a potent instrument for detecting and locating previously unknown network attacks. In this case, the proposed model can be used to detect Heartbleed attacks with good results, as demonstrated in the previous tests. It can also identify if there are new attacks with the same characteristics as Heartbleed hidden from the IDS signature-based because there is no signature yet. This is an example, but the same method can be used to find if there is anomalous traffic generated by infiltration on the network or if the system is under a new type of DoS or Bruteforce attack, and so on.

There are many ways one could extend this thesis in the future. This proposed model can be improved to get better results during the testing phase, adding more layers and neurons to the structure. The proposed model can be evaluated using other datasets or a combination of datasets containing as many distinct benign examples as possible. Integration with the CICFlowMeter and TCPdump tool can be developed to collect traffic and analyze it in real-time in a real network used as a test-bed. The proposed model can be used to build an ensemble model, where each model is specialized to recognize a single attack.

Appendix A

Features Datasets

Name	Description
Flow ID	It is the identificator of the flow, it is composed by Src/Dst IP, Src/Dst Port and Timestamp
Src IP	The Source IP of the flow
Src Port	The Source Port of the flow
Dst IP	The Destination IP of the flow
Dst Port	The Destination Port of the flow
Protocol	Protocol used during the flow
Timestamp	Time and date when the flow is created
Flow Duration	Duration of flow in microseconds
Tot Fwd Pkts	Total number of packets in forward direction
Tot Bwd Pkts	Total number of packets in backward direction
TotLen Fwd Pkts	Total size of packets in forward direction
TotLen Bwd Pkt	Total size of packets in backward direction
Fwd Pkt Len Max	Maximum size of packets in forward direction
Fwd Pkt Len Min	Minimum size of packets in forward direction
Fwd Pkt Len Mean	Mean size of packets in forward direction
Fwd Pkt Len Std	Standard size of packets in forward direction
Bwd Pkt Len Max	Maximum size of packets in backward direction
Bwd Pkt Len Min	Minimum size of packets in backward direction
Bwd Pkt Len Mean	Mean size of packets in backward direction
Bwd Pkt Len Std	Standard size of packets in backward direction
Flow Byts/s	Number of bytes of flow per second
Flow Pkts/s	Number of flow packets per second
Flow IAT Mean	Average time between two packets sent in the flow
Flow IAT Std	Standard time between two packets sent in the flow
Flow IAT Max	Maximum time between two packets sent in the flow
Flow IAT Min	Minimum time between two packets sent in the flow
Fwd IAT Tot	Total of the variation time between two packets sent in the forward direction
Fwd IAT Mean	Mean of the variation time between two packets sent in the forward direction
Fwd IAT Std	Standard of the variation time between two packets sent in the forward direction

Fwd IAT Max	Maximum of the variation time between two packets sent in the forward direction
Fwd IAT Min	Minimum of the variation time between two packets sent in the forward direction
Bwd IAT Tot	Total of the variation time between two packets sent in the backward direction
Bwd IAT Mean	Mean of the variation time between two packets sent in the backward direction
Bwd IAT Std	Standard of the variation time between two packets sent in the backward direction
Bwd IAT Max	Maximum of the variation time between two packets sent in the backward direction
Bwd IAT Min	Minimum of the variation time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag has been set in forward packets (it is 0 for UDP)
Bwd PSH Flags	Number of times the PSH flag has been set in backward packets (it is 0 for UDP)
Fwd URG Flags	Number of times the URG flag has been set in forward packets (it is 0 for UDP)
Bwd URG Flags	Number of times the URG flag has been set in backward packets (it is 0 for UDP)
Fwd Header Len	Total bytes used for header in forward direction
Bwd Header Len	Total bytes used for header in backward direction
Fwd Pkts/s	Number of packets per second in the forward direction
Bwd Pkts/s	Number of packets per second in the backward direction
Pkt Len Min	Minimum length of a packets (payload + header), it is the MTU (Maximum Transmission Unit)
Pkt Len Max	Maximum length of a packet (payload + header), it is the MTU (Maximum Transmission Unit)
Pkt Len Mean	Mean length of a packet (payload + header), it is the MTU (Maximum Transmission Unit)
Pkt Len Std	Standard length of a packet (payload + header), it is the MTU (Maximum Transmission Unit)
Pkt Len Var	Variance length of a packet (payload + header), it is the MTU (Maximum Transmission Unit)
FIN Flag Cnt	Total number of packets with FIN
SYN Flag Cnt	Total number of packets with SYN
RST Flag Cnt	Total number of packets with RST
PSH Flag Cnt	Total number of packets with PSH
ACK Flag Cnt	Total number of packets with ACK
URG Flag Cnt	Total number of packets with URG
CWE Flag Count	Total number of packets with CWE
ECE Flag Cnt	Total number of packets with ECE
Down/Up Ratio	Download and upload report
Pkt Size Avg	Average packet size (payload only), it is equal to MSS (Maximum Segment Size)
Fwd Seg Size Avg	Average packet size (payload only) in the forward direction, it is equal to MSS (Maximum Segment Size)

Bwd Seg Size Avg	Average packet size (payload only) in the backward direction, it is equal to MSS (Maximum Segment Size)
Fwd Byts/b Avg	Average number of byte bulk rates in the forward direction
Fwd Pkts/b Avg	Average number of packet bulk rates in the forward direction
Fwd Blk Rate Avg	Average number of bulk rates in forward direction
Bwd Byts/b Avg	Average number of byte bulk rates in the backward direction
Bwd Pkts/b Avg	Average number of packet bulk rates in the backward direction
Bwd Blk Rate Avg	Average number of bulk rates in backward direction
Subflow Fwd Pkts	The average number of packets in a secondary flow (same IP, different ports) in forward direction
Subflow Fwd Byts	The average number of bytes in a secondary flow (same IP, different ports) in forward direction
Subflow Bwd Pkts	The average number of packets in a secondary flow (same IP, different ports) in backward direction
Subflow Bwd Byts	The average number of bytes in a secondary flow (same IP, different ports) in backward direction
Init Fwd Win Byts	The total number of bytes sent in the initial window in forward direction
Init Bwd Win Byts	The total number of bytes sent in the initial window in backward direcftion
Fwd Act Data Pkts	Packet count with at least 1 byte in TCP payload in forward direction
Fwd Seg Size Min	Minimum size of segments in the Forward Direction
Active Mean	Mean time of an active stream before becoming inactive
Active Std	Standard time of an active stream before becoming inactive
Active Max	Maximum time of an active stream before becoming inactive
Active Min	Minimum time of an active stream before becoming inactive
Idle Mean	Mean idle time of a flow before activating
Idle Std	Standard idle time of a flow before activating
Idle Max	Maximum idle time of a flow before activating
Idle Min	Minimum idle time of a flow before activating
Label	Classification of the flow (Benign, DDoS, DoS Hulk, ...)

Table A.1: In this table are listed all the features of the datasets with its description, created with the CICFlowMeter tool. Source: https://github.com/ahlashkari/CICFlowMeter

Appendix B

Histograms

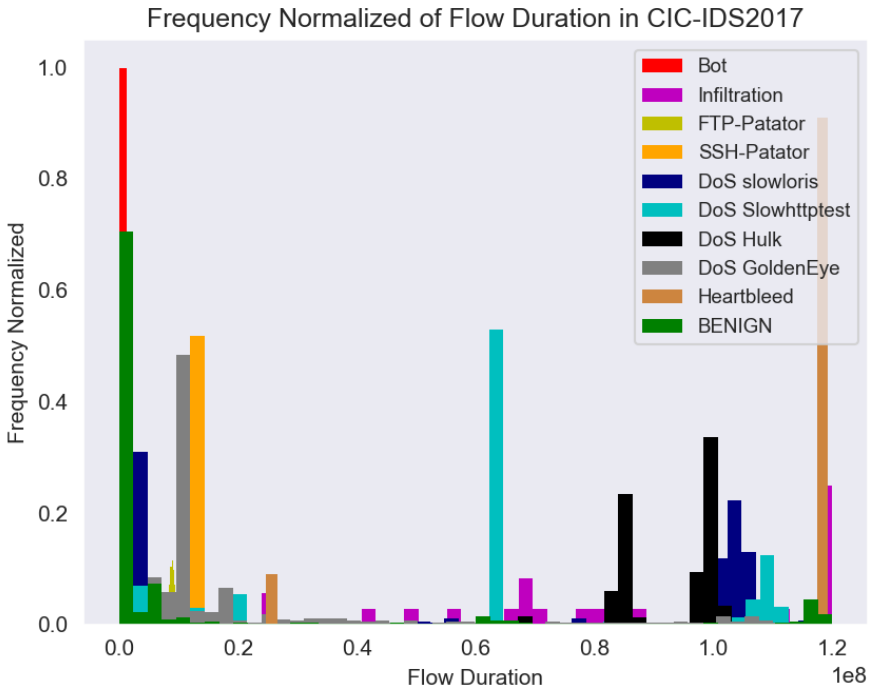


Figure B.1: Histograms for CIC-IDS2017 dataset attacks for the feature "Flow Duration".

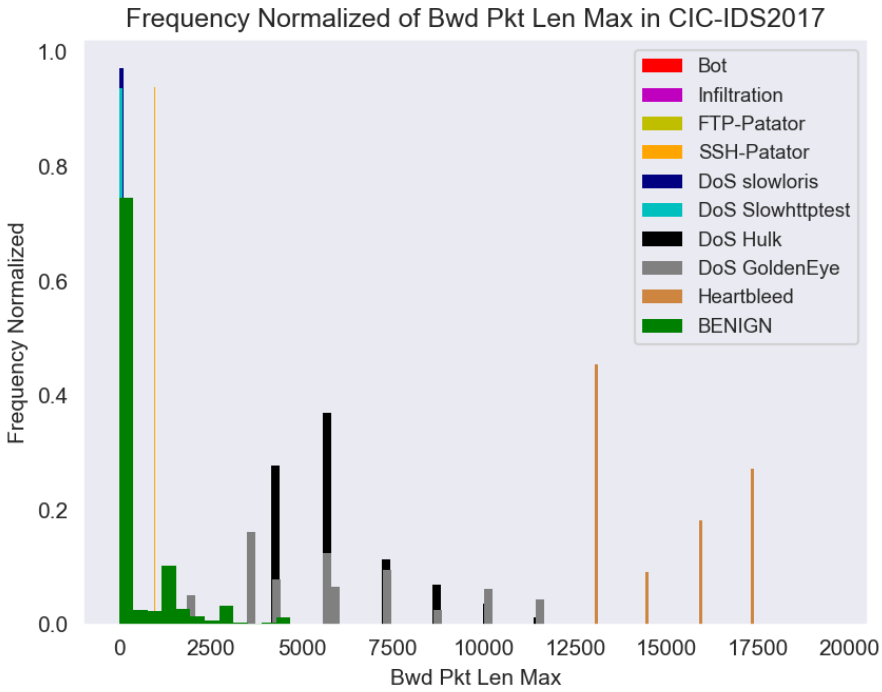


Figure B.2: Histograms for CIC-IDS2017 dataset attacks for the feature "Bwd Pkt Len Max".

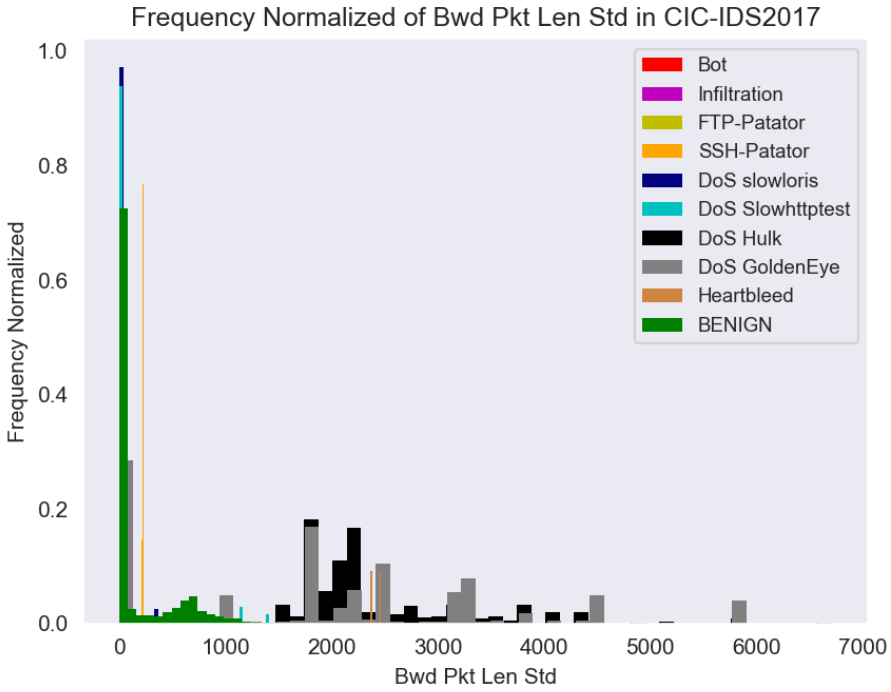


Figure B.3: Histograms for CIC-IDS2017 dataset attacks for the feature "Bwd Pkt Len Std".

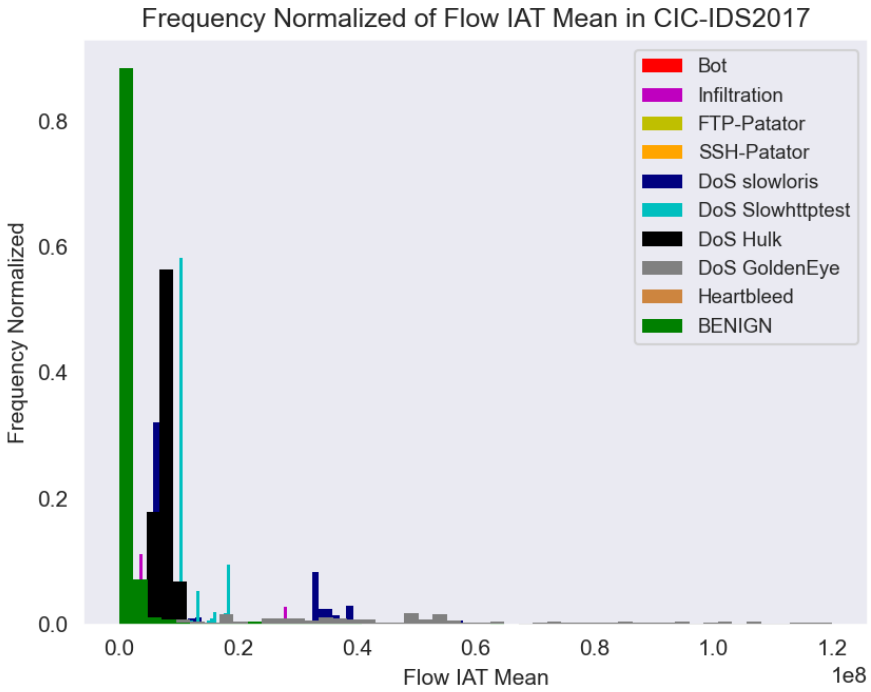


Figure B.4: Histograms for CIC-IDS2017 dataset attacks for the feature "Flow IAT Mean".

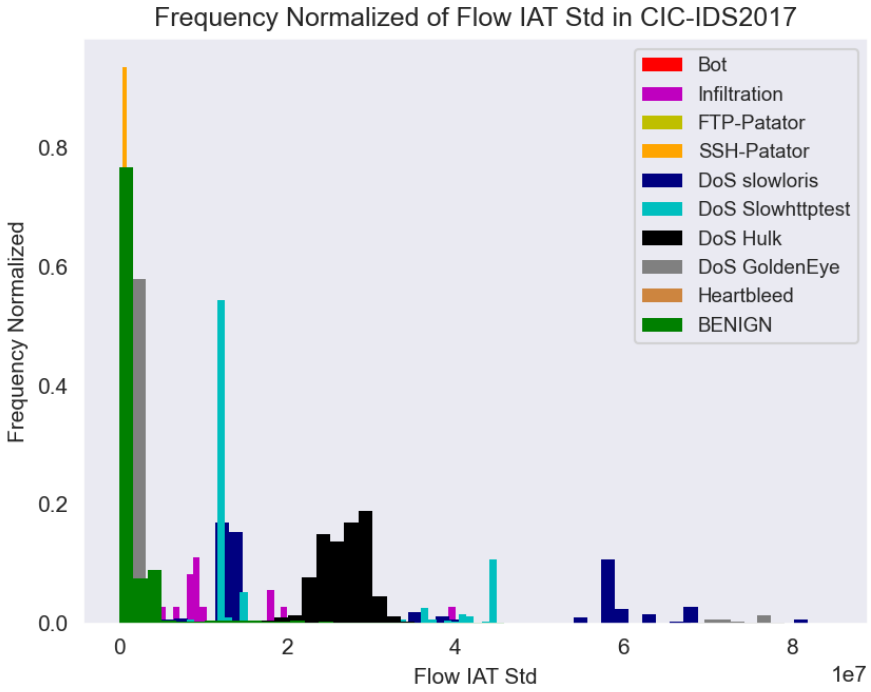


Figure B.5: Histograms for CIC-IDS2017 dataset attacks for the feature "Flow IAT Std".

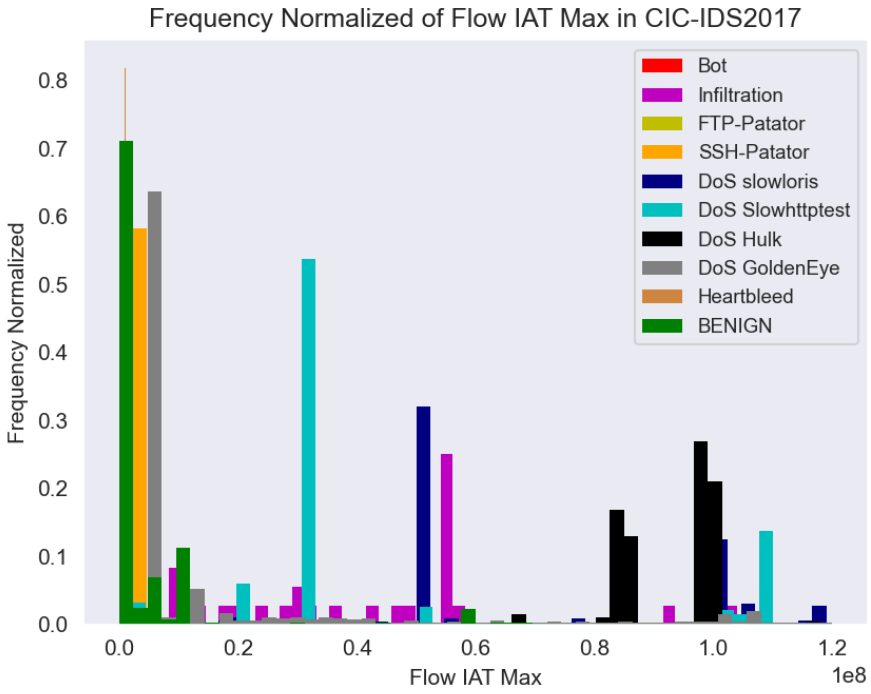


Figure B.6: Histograms for CIC-IDS2017 dataset attacks for the feature "Flow IAT Max".

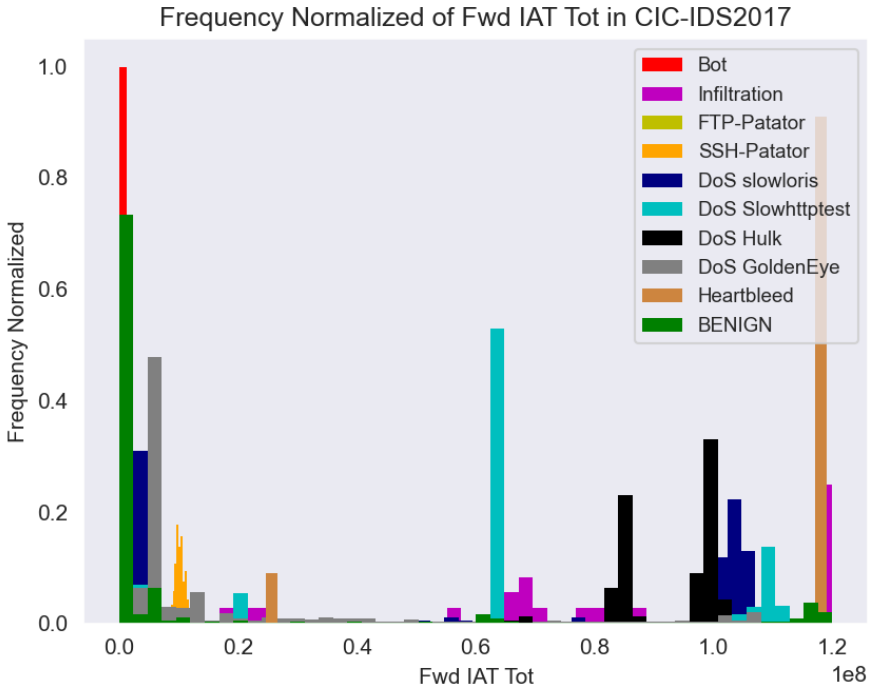


Figure B.7: Histograms for CIC-IDS2017 dataset attacks for the feature "Fwd IAT Tot".

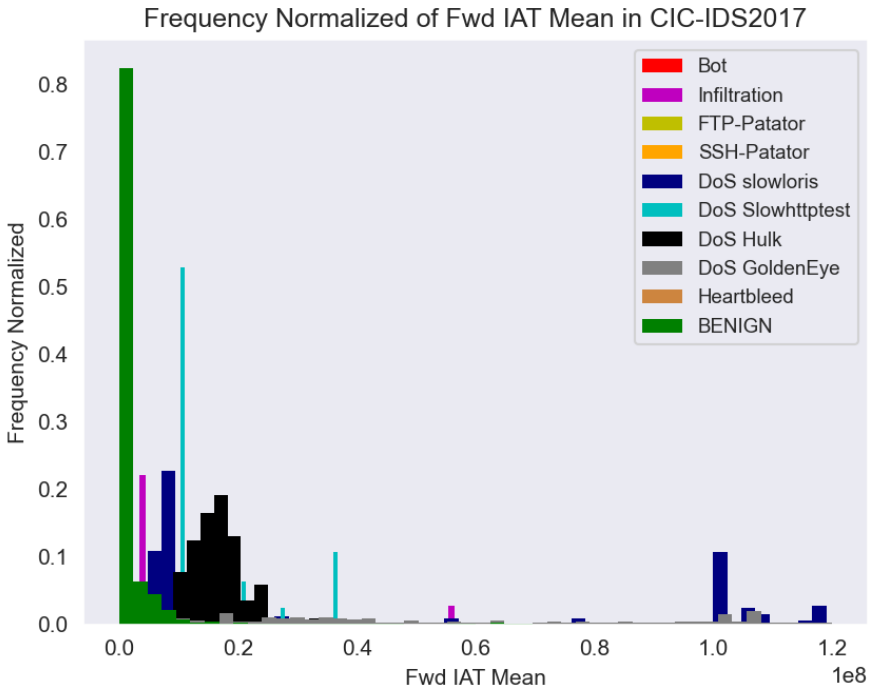


Figure B.8: Histograms for CIC-IDS2017 dataset attacks for the feature "Fwd IAT Mean".

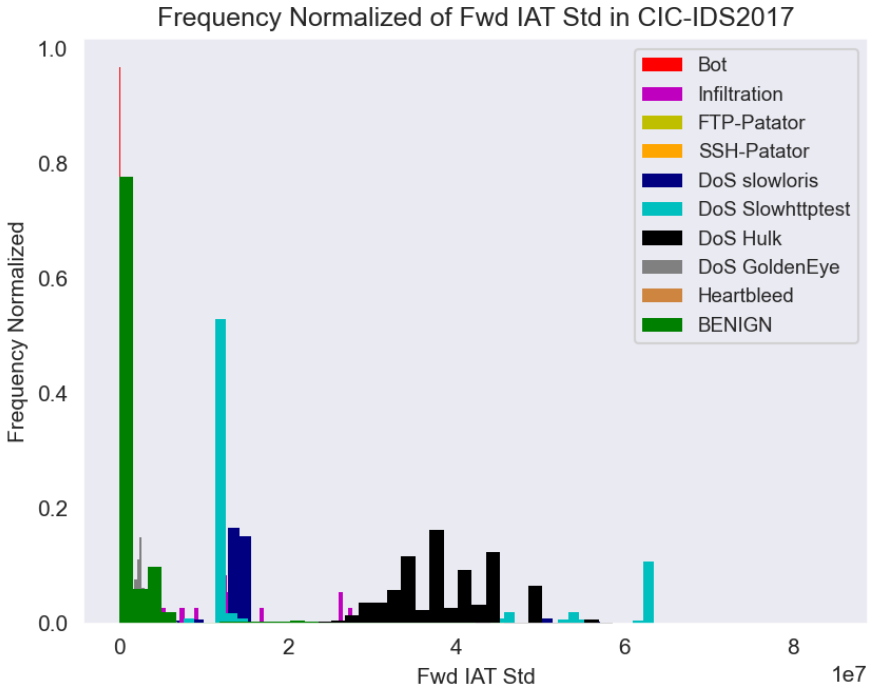


Figure B.9: Histograms for CIC-IDS2017 dataset attacks for the feature "Fwd IAT Std".

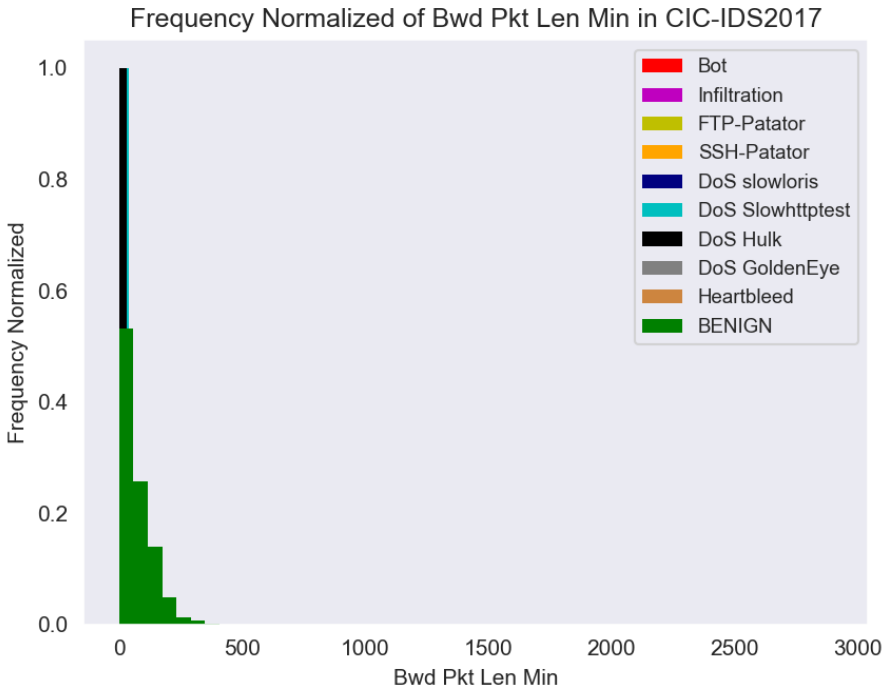


Figure B.10: Histograms for CIC-IDS2017 dataset attacks for the feature "Bwd Pkt Len Min".

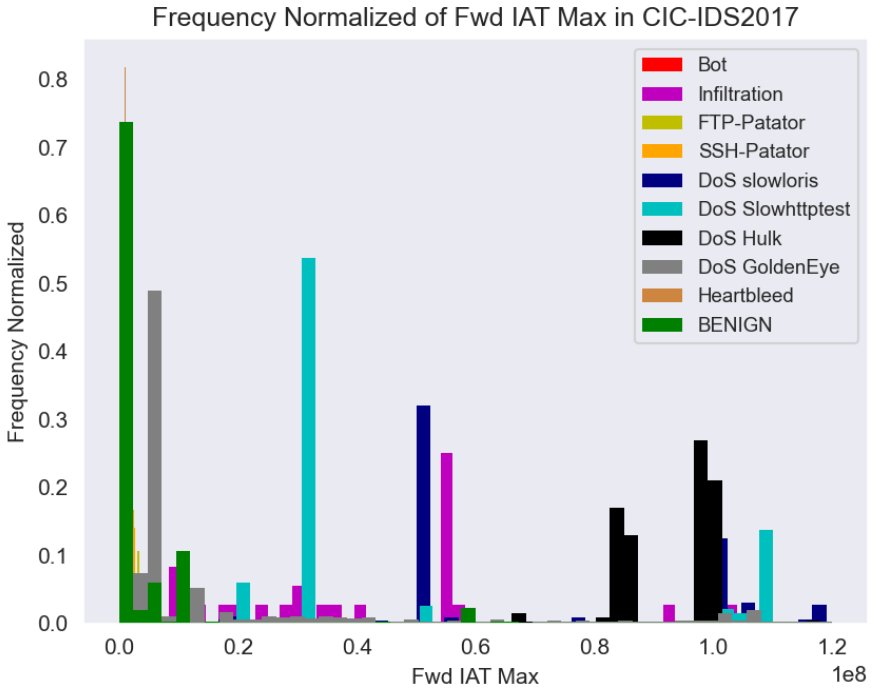


Figure B.11: Histograms for CIC-IDS2017 dataset attacks for the feature "Fwd IAT Max".

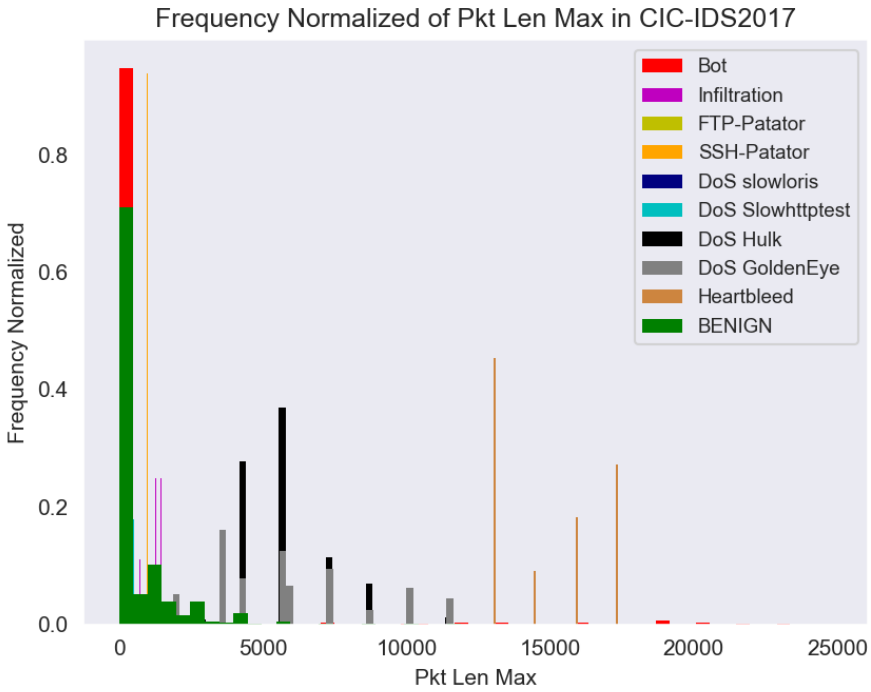


Figure B.12: Histograms for CIC-IDS2017 dataset attacks for the feature "Pkt Len Max".

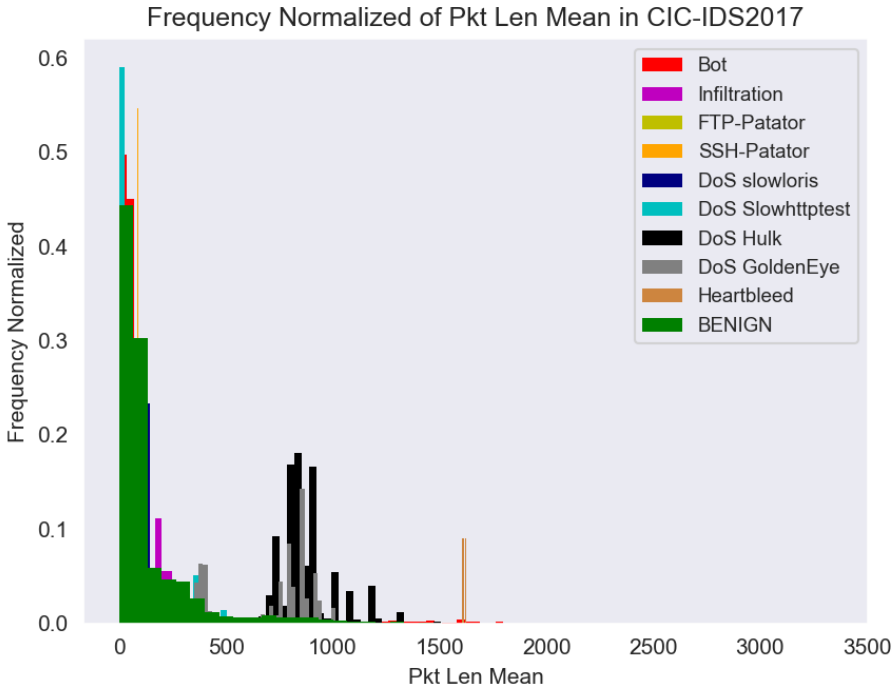


Figure B.13: Histograms for CIC-IDS2017 dataset attacks for the feature "Pkt Len Mean".

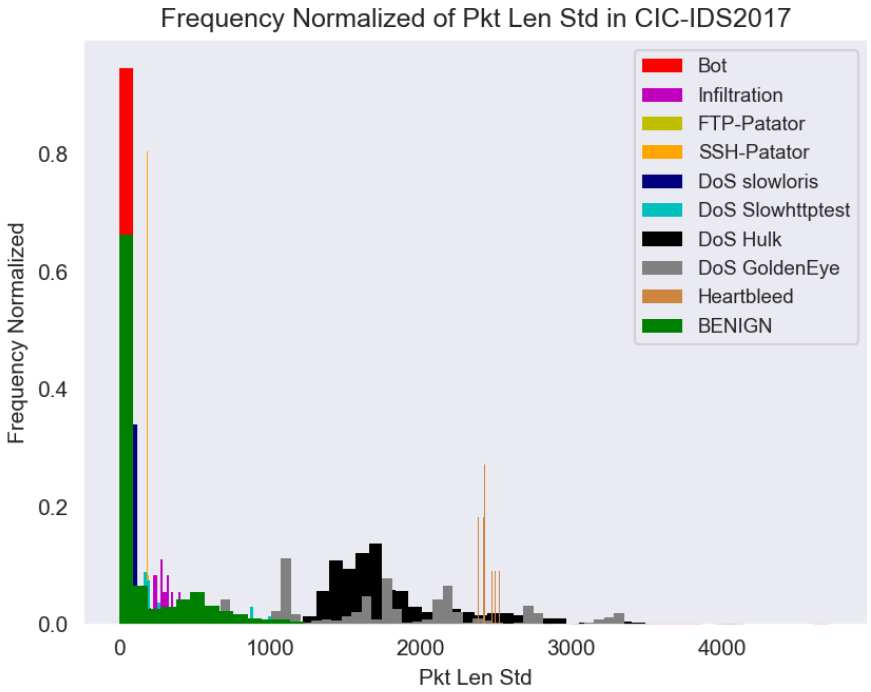


Figure B.14: Histograms for CIC-IDS2017 dataset attacks for the feature “Pkt Len Std”.

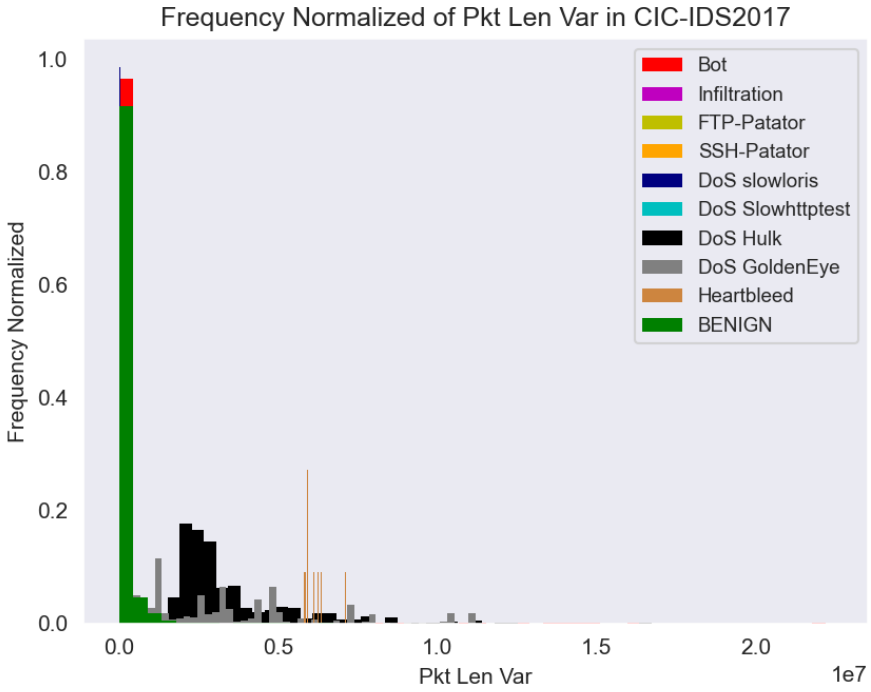


Figure B.15: Histograms for CIC-IDS2017 dataset attacks for the feature “Pkt Len Var”.

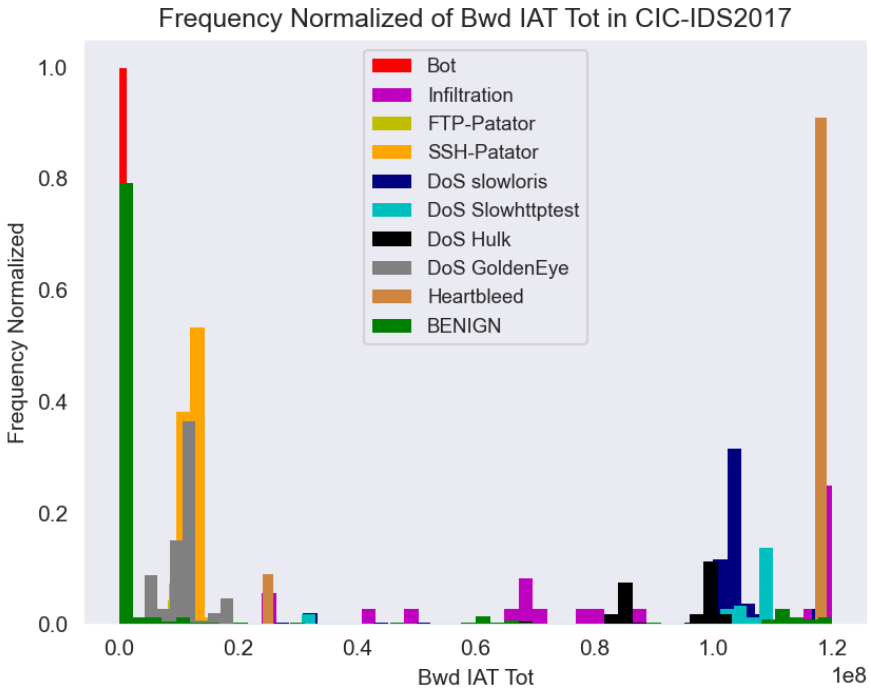


Figure B.16: Histograms for CIC-IDS2017 dataset attacks for the feature "Bwd IAT Tot".

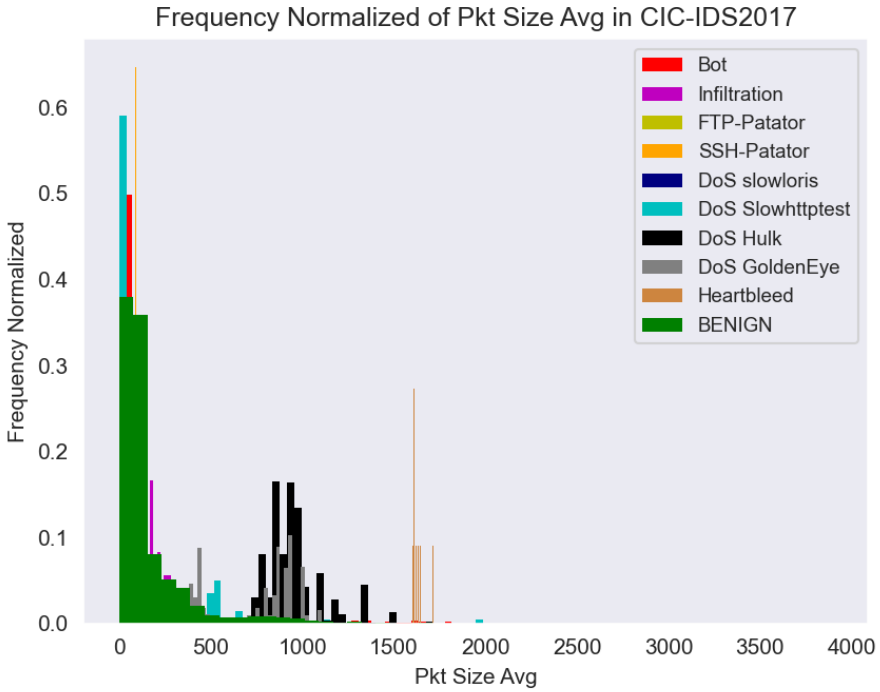


Figure B.17: Histograms for CIC-IDS2017 dataset attacks for the feature "Pkt Size Avg".

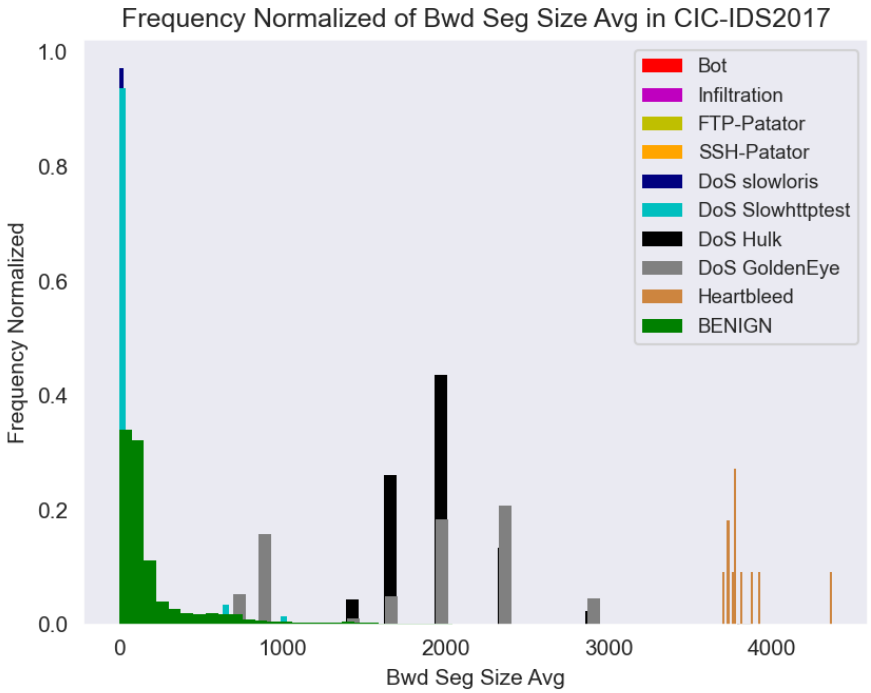


Figure B.18: Histograms for CIC-IDS2017 dataset attacks for the feature "Bwd Seg Size Avg".

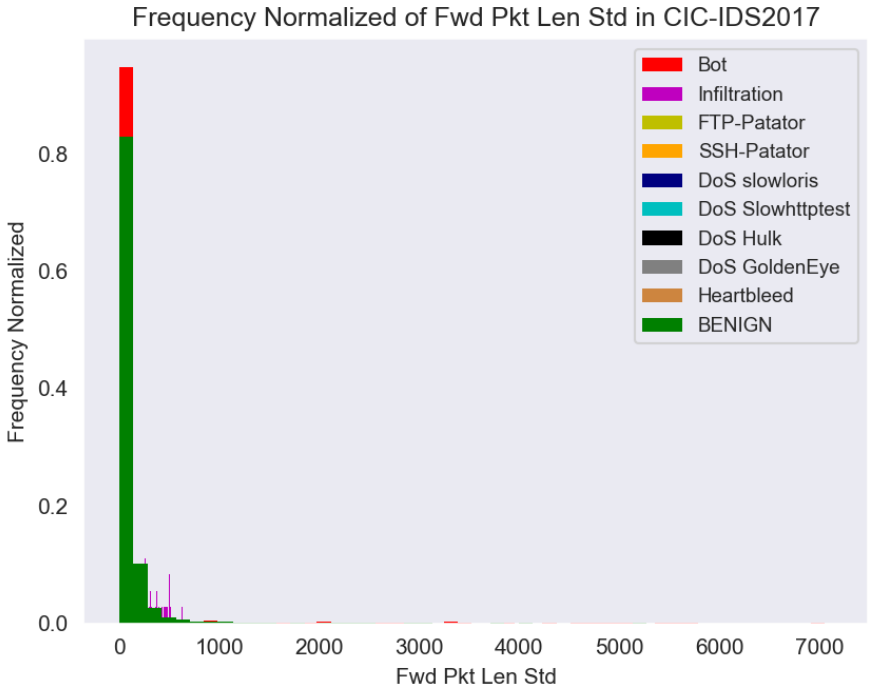


Figure B.19: Histograms for CIC-IDS2017 dataset attacks for the feature "Fwd Pkt Len Std".

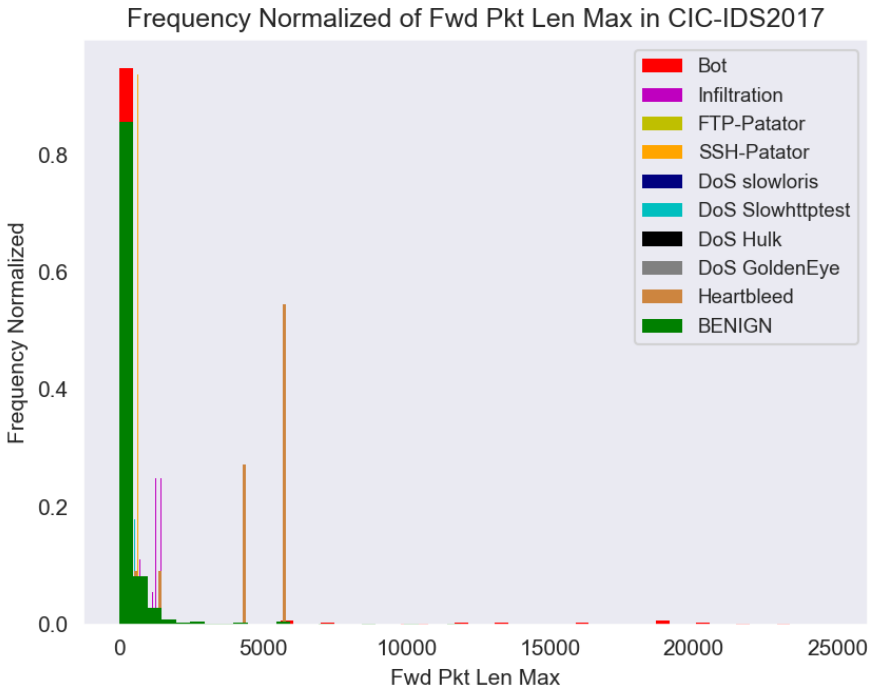


Figure B.20: Histograms for CIC-IDS2017 dataset attacks for the feature "Fwd Pkt Len Max".

Appendix C

User Manual

This Appendix explains the steps for installing and using the software created for this thesis.

C.1 Prerequisites

The software is written in Python 3.9; for installation and managing dependencies, you need python pip3.

It is recommended to use Ubuntu, with at least 8GB of dedicated RAM and 20 GB of free space. To increase the performance, installing the software using Cuda-toolkit[1] and Conda[2] is suggested. Cuda can only run with the use of a supported GPU[3].

C.2 Installing

In this section are explained all the steps performed to install the software.

Ubuntu

Before performing any operation, make sure the package list is up to date with:

```
sudo apt-get update
```

C.2.1 Python 3.9

Ubuntu

You can use the apt package manager using the following command:

```
sudo apt-get install python3.9
```

[1] https://developer.nvidia.com/cuda-toolkit

[2] https://conda-forge.org/

[3] https://developer.nvidia.com/cuda-gpus

Once the installation is complete, use the following command to verify that the installation of python3.9 is successful.

```
python3 --version
```

Windows

To install Python 3.9 for Windows, download the binary file from the site https://www.python.org/downloads/, selecting python3.9. Then, inserts the path where python 3.9 has been installed into the system variables by:

1. Right-clicking *This PC* and select *Properties*.
2. Click on *Advanced system settings*.
3. Click on *Environment Variables*.
4. In *System variables*, select the variable *Path* and click on *Edit*.
5. Click on *New* and insert the Python install directory.

To verify the python3.9 installation, use the following command in a cmd terminal.

```
python3 --version
```

C.2.2 Pip3

Ubuntu

To install pip3 package manager, you need to run the following command:

```
apt-get install python3-pip
```

Windows

The pip3 package manager is already included on systems that have installed python 3.9 via the executable.

C.2.3 Conda and Cuda-toolkit (only supported GPUs)

These steps are for using Conda and Cuda-toolkit to increase the performance of the software. Do not do these steps unless you have a supported GPU[4]. If If you donât have a GPU that supports Cuda-toolkit and Conda, skip this step.

[4] https://developer.nvidia.com/cuda-gpus

Ubuntu

First, install Conda you can choose to install Miniconda https://docs.conda.io/en/latest/miniconda.html#linux-installers or Anaconda https://www.anaconda.com/products/distribution. Remember to choose the correct Python 3.9 version. Miniconda is recommended. Execute the executable file with:

```
bash Miniconda3-latest-Linux-x86_64.sh
```

To enable the command `conda`, you should restart the terminal or execute the command:

```
source  /.bashrc
```

To verify the successful installation of Conda, execute the following command:

```
conda -V
```

Check the NVIDIA driver with the following command:

```
nvidia-smi
```

Then, install the cuda-toolkit. You need to select the version of cuda-toolkit supported by your GPU. You can use `nvcc`[5] to discover it.

```
conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0
```

Configure the system path to the variable of LD_LIBRARY with the following command:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/
```

Windows

The current Tensorflow version 2.9.1 it is not supported on the native Windows GPU. You will need to install WSL[6] and then CUDA on WSL[7]. For Conda, you can install the executable of Anaconda https://www.anaconda.com/products/distribution or Miniconda https://docs.conda.io/en/latest/miniconda.html#windows-installers. Remember to choose the correct Python 3.9 version.

C.2.4 Dependencies

In both software, there is a file called *requirements.txt*, which contains the different python modules necessary for the software.

This is done after the previous steps of installing python3.9 and pip3. For both operating systems (Ubuntu and Windows) you need to open a terminal in the software folder and run the following command:

```
pip install -r requirements.txt
```

[5] https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html

[6] https://learn.microsoft.com/it-it/windows/wsl/install

[7] https://docs.nvidia.com/cuda/wsl-user-guide/index.html

C.3 Usage

To run the software with the different options, you must use the following command:

`python3 ./bin/model/main.py -h [-cpu NUM_CPU] [-fh | -rf-xgb | -ae] [-s] [-t] [-t-tor] [-t1] [-t2] [-t3] [-t4]`

At least one of the parameters of `-fh` or `-rf-xgb` or `-ae` must always be present, while the other parameters are optional. Below are explained the different options:

`-h, --help`: displays the help message.

`-cpu NUM_CPU, --cpu NUM_CPU`: insert the number of CPUs (NUM_CPU) used to open the datasets. Note that to insert all the CPUs of the system. The default value is 2.

`-fh, --features-histogram`: generate histograms for each dataset feature. this parameter displays the histogram only; to also save them in the folder `/bin/histograms`, you must also enter the parameter `-s`.

`-s, --save-histogram`: save the histograms generated with the parameter `-fh, --features-histogram` in the folder `/bin/histograms`.

`-rf-xgb, --random-forest-extreme-gb`: use Random Forest and Extreme Gradient Boosting. This parameter executes all the training and the testing (test with CIC-IDS2017, the Anomaly Attack Test) of the supervised models. It also generates the plots of the ROC obtained and the confusion matrix for each test and for each model.

`-ae, --auto-encoder`: use the proposed auto-encoder model. This parameter must be followed by the other parameters (`[-t] [-t-tor] [-t1] [-t2] [-t3] [-t4]`), to train and test with different datasets

`-t, --train`: the parameter must be preceded by `-ae, --auto-encoder`. The proposed model is trained with the dataset CIC-IDS2017. The graph of the loss function of the obtained model is generated. The model is saved in the folder `/bin/model/model_save/AE`, and the other functional structures are saved in `/bin/model/model_save/`. Warning: the previously obtained model will be overwritten with this parameter.

`-t-tor, --train-tor`: the parameter must be preceded by `-ae, --auto-encoder`. The proposed model is trained with the dataset TORSEC. The graph of the loss function of the obtained model is generated. The model is saved in the folder `/bin/model/model_save/AE_anomaly_test_benign`, and the other functional structures are saved in `/bin/model/model_save/`. Warning: the previously obtained model will be overwritten with this parameter.

`-t1, --test1`: the parameter must be preceded by `-ae, --auto-encoder`. Test the proposed auto-encoder model trained with the CIC-IDS2017 dataset and tested with the same dataset. It shows the plot of the reconstruction error, the confusion matrix and the ROC obtained for each attack.

`-t2, --test2`: the parameter must be preceded by `-ae, --auto-encoder`. It performs the Anomaly Attack Test. Test the proposed auto-encoder model trained with the CIC-IDS2017 dataset and tested with the TORSEC dataset. It shows the plot of the reconstruction error, the confusion matrix and the ROC obtained for each attack.

`-t3, --test3`: the parameter must be preceded by `-ae, --auto-encoder`. It performs the Anomaly Attack Test. Test the proposed auto-encoder model trained with the TORSEC dataset and tested with the TORSEC dataset. It shows the plot of the reconstruction error, the confusion matrix and the ROC obtained for each attack.

`-t4, --test4`: the parameter must be preceded by `-ae, --auto-encoder`. It tests the proposed auto-encoder model with the Heartbleed dataset. It shows the plot of the reconstruction error, the confusion matrix and the ROC obtained for the Heartbleed attack.

C.4 Jupyter notebook

The software is also available with Jupyter[8] notebooks, which can help you better understand the code, as each piece of code is run individually and is displayed under it.

C.4.1 Installation

It is recommended to use DataSpell[9] by JetBrains or using JupiterLab, to open Jupyter notebooks. To install JupiterLab:

```
pip install jupyterlab
```

C.4.2 Usage

To run JupiterLab, use the following command:

```
jupyter-lab
```

Then navigate to the software directory and open the files `.ipynb`. After selecting a file, for example, autoencoder.ipynb, this will show you different parts of code in other blocks. Under each block is his execution.

[8] https://jupyter.org/

[9] https://www.jetbrains.com/dataspell/

Appendix D

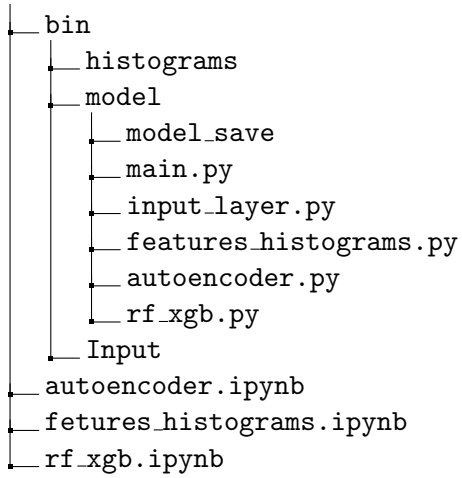
Developer Manual

D.1 The Proposed Auto-Encoder Model

This section describes the structure of the software developed in this thesis and how to change the dataset and add new tests.

D.1.1 Structure

The structure of the software developed in this thesis is as follows:



In this project, there are two different files: the python files with extension `.py` that are in the path `/bin/model` and the jupyter files with the extension `.ipynb` which are in the root directory. These two files have the same code; the difference is that the jupyter files contain the code with its execution below.

Inside the directory `/bin`, there are three main directories: `histograms`, `model` and `Input`. The directory `histograms` has all the histograms generated for the selected dataset (i.e. CIC-IDS2017). Instead, the `model` has all the source code of the model written in `python3.9` and the already trained models inside the directory `model_save`. The `Input` directory contains all the datasets used in this thesis.

D.1.2 Source Files

main.py

The file `main.py` is the initiator of the entire program. From here all the functions of the other files are called according to the parameter passed by the command line. In the `main.py` file, there are three fundamental parameters:

- num_processors: the number of CPUs used to open the dataset. The default value is 2. This value can be set from the command line, or you can change the standard value and put it at will without passing it from the command line.
- path_name_cic17: it is the path of the CIC-IDS2017 dataset. The default path is `/bin/Input/cic17`.
- path_name_torsec: it is the path of the TORSEC dataset. The default path is `/bin/Input/TORSEC/Converted_CIC`.

input_layer.py

It contains the function `openDatasetMultiprocessor(path_save, colum_names2, num_process)`, which allows you to open any dataset in multiprocessing. This requires the path of the dataset in `.csv` to open (`path_save`), the names of the column in the dataset (`colum_names2`) and the number of CPUs used to open the dataset. The `path_save` could be a single `.csv` (e.g `/Input/dataset.csv`) or a group of `.csv` (e.g `/Input/*.csv`). Then it returns the opened dataset in `pandas`.

features_histograms.py

The main function is `fetures_histograms(num_process, save_histogram, path_name)` that is used to generate the histograms. This function receives the number of CPUs used to open the dataset (`num_process`), the flag `save_histogram`, which indicates if the histogram should be saved in `/bin/histograms` or not, and the path of the CIC-IDS2017 dataset (`path_name`).

autoencoder.py

Here is the source code of the proposed auto-encoder model. the primary function is `proposed_model_call(num_process, path_name_cic17, path_name_torsec, train, train_tor, test1, test2, test3, test4)` which receives: the number of the CPUs used to open datasets, the path of the CIC-IDS2017 and TORSEC dataset and then the flags for training the model with the corresponding dataset (`train, train_tor`) or testing it (`test1, test2, test3, test4`). For each test, there is a `if` section of the relative test executed on the model. If you want to add a new test, you can add it here by setting the relative flag and loading the relative model with `keras.models.load_model(r'./model_save/AE')`, indicating the path of where the model is located.

The proposed model is built and trained with the function `proposed_model_train(x_train, batch_size, name_model_to_save)`. It receives the dataset (`x_train`), the `batch_size` and where to save the model (`name_model_to_save`), this indicates the folder

inside the directory `/bin/model/model_save`. In the first part, you can change the hyperparameters of the model and its structure.

In the first part of the `proposed_model_call`, there is the path where the scaler and the encoder are saved and loaded, for both the CIC-IDS2017 and TORSEC datasets. In the default case, they are saved in the directory `/bin/model/model_save`.

rf_xgb.py

The primary function is `rf_xgb(num_process, path_name_cic17, path_name_torsec)` which receives the number of CPUs used to open the dataset, the path of the datasets CIC-IDS2017 and TORSEC. All the tests performed are handled in this function.

Bibliography

[1] U. Sabeel, S. S. Heydari, K. Elgazzar, and K. El-Khatib, "Building an intrusion detection system to detect atypical cyberattack flows", IEEE Access, vol. 9, 2021, pp. 94352–94370, DOI 10.1109/ACCESS.2021.3093830

[2] S. Zavrak and M. Iskefiyeli, "Anomaly-based intrusion detection from network flow features using variational autoencoder", IEEE Access, vol. 8, 2020, pp. 108346–108358, DOI 10.1109/ACCESS.2020.3001350

[3] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization", 01 2018, pp. 108–116, DOI 10.5220/0006639801080116

[4] N. Gupta, V. Jindal, and P. Bedi, "Cse-ids: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems", Computers Security, vol. 112, 2022, p. 102499, DOI 10.1016/j.cose.2021.102499

[5] P. Bedi, N. Gupta, and V. Jindal, "Siam-ids: Handling class imbalance problem in intrusion detection systems using siamese neural network", Procedia Computer Science, vol. 171, 2020, pp. 780–789, DOI 10.1016/j.procs.2020.04.085. Third International Conference on Computing and Network Communications (CoCoNet'19)

[6] P. Bedi, N. Gupta, and V. Jindal, "I-siamids: an improved siam-ids for handling class imbalance in network-based intrusion detection systems", CoRR, vol. abs/2009.10940, 2020, DOI 10.48550/arXiv.2009.10940

[7] N. Gupta, V. Jindal, and P. Bedi, "Lio-ids: Handling class imbalance using lstm and improved one-vs-one technique in intrusion detection system", Computer Networks, vol. 192, 2021, p. 108076, DOI 10.1016/j.comnet.2021.108076

[8] T. Zoppi, M. Gharib, M. Atif, and A. Bondavalli, "Meta-learning to improve unsupervised intrusion detection in cyber-physical systems", ACM Trans. Cyber-Phys. Syst., vol. 5, sep 2021, DOI 10.1145/3467470

[9] V. Hautamaki, I. Karkkainen, and P. Franti, "Outlier detection using k-nearest neighbour graph", Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004., 2004, pp. 430–433 Vol.3, DOI 10.1109/ICPR.2004.1334558

[10] H.-P. Kriegel, M. Schubert, and A. Zimek, "Angle-based outlier detection in high-dimensional data", Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2008, pp. 444–452, DOI 10.1145/1401890.1401946

[11] F. Iglesias VÃ¡zquez, T. Zseby, and A. Zimek, "Outlier detection based on low density models", 2018 IEEE International Conference on Data Mining Workshops (ICDMW), 2018, pp. 970–979, DOI 10.1109/ICDMW.2018.00140

[12] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", Proceedings of the Second

International Conference on Knowledge Discovery and Data Mining, 1996, pp. 226–231

[13] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study", Journal of Information Security and Applications, vol. 50, 2020, p. 102419, DOI 10.1016/j.jisa.2019.102419

[14] F. S. L. Filho, F. A. F. Silveira, A. de Medeiros Brito Junior, G., S. Vargas-Solar, and L. F. Silveira., "Smart detection: An online approach for dos/ddos attack detection using machine learning", Security and Communication Networks, vol. 2019, Oct 2019, p. 1574749, DOI 10.1155/2019/1574749

[15] J. Kurose and K. Ross, "Computer networking: A top-down approach, global edition", Pearson Education, 2018, ISBN: 9781292153605

[16] "Internet Protocol." RFC 791, September 1981, DOI 10.17487/RFC0791

[17] D. Murray, T. Koziniec, K. Lee, and M. Dixon, "Large mtus and internet performance", 2012 IEEE 13th International Conference on High Performance Switching and Routing, 2012, pp. 82–87, DOI 10.1109/HPSR.2012.6260832

[18] Y. Rekhter and T. Li, "An Architecture for IP Address Allocation with CIDR." RFC 1518, September 1993, DOI 10.17487/RFC1518

[19] "Transmission Control Protocol." RFC 793, September 1981, DOI 10.17487/RFC0793

[20] "User Datagram Protocol." RFC 768, August 1980, DOI 10.17487/RFC0768

[21] T. Polk and S. Turner, "Prohibiting Secure Sockets Layer (SSL) Version 2.0." RFC 6176, March 2011, DOI 10.17487/RFC6176

[22] R. Barnes, M. T., A. Pironti, and A. Langley, "Deprecating Secure Sockets Layer Version 3.0." RFC 7568, June 2015, DOI 10.17487/RFC7568

[23] C. Allen and T. Dierks, "The TLS Protocol Version 1.0." RFC 2246, January 1999, DOI 10.17487/RFC2246

[24] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1." RFC 4346, April 2006, DOI 10.17487/RFC4346

[25] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246, August 2008, DOI 10.17487/RFC5246

[26] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, August 2018, DOI 10.17487/RFC8446

[27] M. Schneider, H. Shulman, A. Sidis, R. Sidis, and M. Waidner, "Diving into email bomb attack", 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2020, pp. 286–293, DOI 10.1109/DSN48063.2020.00045

[28] D. Yuan and J. Zhong, "A lab implementation of syn flood attack and defense", Proceedings of the 9th ACM SIGITE Conference on Information Technology Education, New York, NY, USA, 2008, pp. 57–58, DOI 10.1145/1414558.1414575

[29] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: a classification", Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology (IEEE Cat. No.03EX795), 2003, pp. 190–193, DOI 10.1109/ISSPIT.2003.1341092

[30] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities", IEEE Communications Surveys Tutorials, vol. 21, no. 2, 2019, pp. 1851–1877, DOI 10.1109/COMST.2019.2891891

[31] M. Williams, M. Tuxen, and R. Seggelmann, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension." RFC 6520, February 2012, DOI 10.17487/RFC6520

[32] A. Nursetyo, D. R. Ignatius Moses Setiadi, E. H. Rachmawanto, and C. A. Sari, "Website and network security techniques against brute force attacks using honeypot", 2019 Fourth International Conference on Informatics and Computing (ICIC), 2019, pp. 1–6, DOI 10.1109/ICIC47613.2019.8985686

[33] L. Bošnjak, J. Sreš, and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords", 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2018, pp. 1161–1166, DOI 10.23919/MIPRO.2018.8400211

[34] H. Kumar, S. Kumar, R. Joseph, D. Kumar, S. K. Shrinarayan Singh, A. Kumar, and P. Kumar, "Rainbow table to crack password using md5 hashing algorithm", 2013 IEEE Conference on Information Communication Technologies, 2013, pp. 433–439, DOI 10.1109/CICT.2013.6558135

[35] M. Himelein-Wachowiak, S. Giorgi, A. Devoto, M. Rahman, L. Ungar, H. A. Schwartz, D. H. Epstein, L. Leggio, and B. Curtis, "Bots and misinformation spread on social media: Implications for COVID-19", J. Med. Internet Res., vol. 23, May 2021, p. e26933, DOI 10.2196/26933

[36] M. Ozkan-Okay, R. Samet, Ö. Aslan, and D. Gupta, "A comprehensive systematic literature review on intrusion detection systems", IEEE Access, vol. 9, 2021, pp. 157727–157760, DOI 10.1109/ACCESS.2021.3129336

[37] X. Zhang, C. Li, and W. Zheng, "Intrusion prevention system design", The Fourth International Conference onComputer and Information Technology, 2004. CIT '04., 2004, pp. 386–390, DOI 10.1109/CIT.2004.1357226

[38] N. Technology, "Nist sp 800-94 - guide to intrusion detection and prevention systems (idps)", CreateSpace Independent Publishing Platform, 2007, ISBN: 9781547257225

[39] P. Ongsulee, "Artificial intelligence, machine learning and deep learning", 2017 15th International Conference on ICT and Knowledge Engineering (ICTKE), 2017, pp. 1–6, DOI 10.1109/ICTKE.2017.8259629

[40] "Glossary of terms", Machine Learning, vol. 30, Feb 1998, pp. 271–274, DOI 10.1023/A:1017181826899

[41] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications", ACM Comput. Surv., vol. 51, sep 2018, DOI 10.1145/3234150

[42] V. Kotu and B. Deshpande, "Chapter 2 - data mining process", Data Science (Second Edition) (V. Kotu and B. Deshpande, eds.), pp. 19–37, Morgan Kaufmann, second edition ed., 2019, DOI 10.1016/B978-0-12-814761-0.00002-2

[43] B. D. Ripley, "Glossary", Cambridge University Press, 1996, ISBN: 978-0-52-171770-0

[44] T. Iliou, C. N. Anagnostopoulos, M. Nerantzaki, and G. Anastassopoulos, "A novel machine learning data preprocessing method for enhancing classification algorithms performance", Proceedings of the 16th International Conference on Engineering Applications of Neural Networks (INNS), New York, NY, USA, 2015, DOI 10.1145/2797143.2797155

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", Journal of Machine Learning Research, vol. 12, 2011, pp. 2825–2830, DOI 10.48550/arXiv.1201.0490

[46] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks", Information Processing Management, vol. 45, no. 4, 2009, pp. 427–437, DOI 10.1016/j.ipm.2009.03.002

[47] Z.-H. Zhou, "Machine learning", Springer Nature, 2021, ISBN: 978-981-15-1967-3

[48] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers - a survey", IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 35, no. 4, 2005, pp. 476–487, DOI 10.1109/TSMCC.2004.843247

[49] G. Kyriakides and K. G. Margaritis, "Hands-on ensemble learning with python : Build highly optimized ensemble machine learning models using scikit-learn and keras", Packt Publishing, 2019, ISBN: 978-1789612851

[50] Z.-H. Zhou, "Ensemble methods: foundations and algorithms", ISBN: 978-1439830031

[51] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2016, pp. 785–794, DOI 10.1145/2939672.2939785

[52] S. Pafka, "benchm-ml." `https://github.com/szilard/benchm-ml`, 2019

[53] S. K. Moore, D. Schneider, and E. Strickland, "How deep learning works: Inside the neural networks that power today's ai", IEEE Spectrum, vol. 58, no. 10, 2021, pp. 32–33, DOI 10.1109/MSPEC.2021.9563965

[54] L. Chen, H. Qu, J. Zhao, B. Chen, and J. C. Principe, "Efficient and robust deep learning with correntropy-induced loss function", Neural Computing and Applications, vol. 27, May 2016, pp. 1019–1031, DOI 10.1007/s00521-015-1916-x

[55] Y. Song, S. Hyun, and Y.-G. Cheong, "Analysis of autoencoders for network intrusion detection", Sensors, vol. 21, no. 13, 2021, DOI 10.3390/s21134294

[56] U. Michelucci, "An introduction to autoencoders", CoRR, vol. abs/2201.03898, 2022, DOI 10.48550/arXiv.2201.03898

[57] D. Canavese, L. Regano, C. Basile, G. Ciravegna, and A. Lioy, "Encryption-agnostic classifiers of traffic originators and their application to anomaly detection", Computers Electrical Engineering, vol. 97, 2022, p. 107621, DOI https://doi.org/10.1016/j.compeleceng.2021.107621

[58] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: Supervised or unsupervised?", Image Analysis and Processing – ICIAP 2005 (F. Roli and S. Vitulano, eds.), Berlin, Heidelberg, 2005, pp. 50–57, DOI 10.1007/11553595_6

[59] A. Boukhamla and J. C. Gaviro, "Cicids2017 dataset: performance improvements and validation as a robust intrusion detection system testbed", International Journal of Information and Computer Security, vol. 16, no. 1-2, 2021, pp. 20–32, DOI 10.1504/IJICS.2021.117392

[60] A. Thakkar and R. Lohiya, "A review of the advancement in intrusion detection datasets", Procedia Computer Science, vol. 167, 2020, pp. 636–645, DOI 10.1016/j.procs.2020.03.330

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", 2014, DOI 10.48550/ARXIV.1412.6980

[62] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution", 2010 20th International Conference on Pattern Recognition, 2010, pp. 3121–3124, DOI 10.1109/ICPR.2010.764