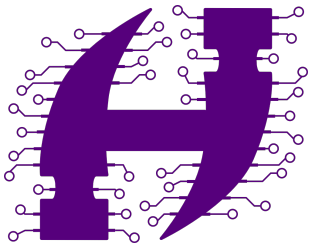


ML for Anomaly Detection in Cyber Security

Testing Datasets for Intrusion Detection using Supervised and Unsupervised Learning Models

Hephaestus Applied Artificial Intelligence Association

Authors:

Member	Role
Antonio Honsell	Project leader
Andrea Lamonarca	Member
Isabella Tasset	Member

Milan, February 16, 2025

Abstract

In this paper we investigate the effectiveness with which cyber-attacks in client-server networks can be detected with the help of machine learning algorithms. Basing ourselves on previous research papers, chiefly Al Nuaimi et al., and taking inspiration from pre-existing datasets, mainly the Edge-IIOT-2022, we pursue to verify their findings and possibly expand them, drawing further conclusions on the applicability of ML methods in Intrusion Detection Systems (IDS) in Cyber Security. To this end, we simulate two types of cyber-attacks, Denial of Service (DOS) and Brute Force (for passwords): we assemble two datasets with the most relevant features and then evaluate the performance of some easily interpretable (supervised and unsupervised) ML algorithms on the binary classification problem of distinguishing between normal and anomalous traffic. In this process, we formulate several hypotheses that are tested throughout the paper, and make some assumptions to simplify the otherwise intractable analysis. Also, we explore the trade-off between performance and privacy by employing the ϵ-differential privacy technique on real datasets.

Contents

1 | Introduction

1.1 | Network Protocols and Cyber Attacks

In this section, we will briefly explain the main concepts necessary for understanding network traffic within the context of our paper, focusing on the Transmission Control Protocol (TCP) and the basic ideas behind the implementation of DOS and Brute Force attacks.
To begin with, **TCP** is a fundamental protocol within the Internet protocol suite, originating from its initial network implementation. It often complements the Internet Protocol (IP) and connection is set up through a procedure known as the three-way handshake. Therefore, once a server is listening on a given port, the client may establish a connection by the following procedure:

1. The client sends a SYN (synchronize request) to the server and sets the sequence number (tcp.seq) for the connection to a random value A.
2. In response, the server replies with a SYN-ACK (synchronize acknowledgment). The acknowledgment number is set to A+1, and the sequence number that the server chooses for the packet is another random number, B.
3. Finally, the client sends an ACK (acknowledgment) back to the server. The sequence number is set to the received acknowledgment value i.e. A+1, and the ACK number is set to B+1.

TCP has become so widespread because applications do not need to manage the specific mechanisms for data transmission between hosts, as TCP handles all the details of the "handshake" as well as of data transmission. The TCP protocol is also able to detect any issues and requests the retransmission of any lost/corrupted data to ensure reliable communication.
However, all of these characteristics of TCP show that the latter protocol is optimized for accurate rather than timely delivery and relatively long delays can thus incur. This makes it particularly vulnerable to **DOS attacks**, which, in most cases (including in the implementation used in our DOS dataset), take advantage of an incomplete three way handshake to lengthen connections and prevent the correct data flow between a server and its clients. More generally, a denial-of-service (DOS) attack constitutes a cyber-attack wherein the attacker aims to render a network unavailable to its intended users by temporarily disrupting the services of a server. This attack is typically executed by inundating the target server with superfluous requests, with the intention of overwhelming the system and thereby preventing the fulfillment of legitimate requests.
TCP is thus essential to effectively analyze network traffic and thus a range of associated features are available to monitor changes in the number of connections, their durations and much more. Such statistics will be recorded in this paper, as in many previous ones, by means of the Wireshark interface, which allows to record traffic on any given local network. The most relevant features offered by Wireshark in the context of TCP, which also mirror the main variables of our datasets and of the Edge-IIot 2022 dataset, are presented below:

- Source TCP Address (tcp.srcport) – The port number of the sender in the packet, indicating where the traffic originated from.
- Destination TCP Address (tcp.dstport) – The port number of the receiver, showing the intended recipient of the packet.
- Sequence Number (tcp.seq) – A unique number assigned to each byte in a TCP connection, ensuring data is reassembled in the correct order.
- Acknowledgment Number (tcp.ack) – The number indicating the next expected byte from the sender, ensuring reliable data delivery.
- TCP Flags (tcp.flags) – A set of control bits (SYN, ACK, FIN, RST) that manage the state of a connection (*e.g.*, initiating, closing, or resetting a connection).
- Window Size (tcp.window_size_value) – The amount of data (in bytes) that a receiver is willing to accept before requiring an acknowledgment.
- TCP Checksum (tcp.checksum) – It is used to verify data integrity and detect transmission errors.
- TCP Length (tcp.len) – The size of the actual data being carried in the TCP segment.

- TCP Stream Index (tcp.stream) – A unique identifier for a TCP session, allowing tracking of packets within the same connection.
- Time Since Last Packet (tcp.time_delta) – The time difference between consecutive packets in the same stream, useful for analyzing traffic delays or anomalies.

Another important protocol we will deal with later is the **SSH** protocol. SSH is a cryptographic network protocol which allows for secure network services over an unsecured network. Its most notable applications are remote login and command-line execution. It is a protocol for secure remote access and command execution over a network.
SSH operates on top of the TCP (Transmission Control Protocol). Specifically SSH runs over TCP port 22 by default. Before SSH communication starts, TCP establishes a connection using the three-way handshake, as already explained previously. After the client and the server have established a TCP connection (TCP handshake), then the SSH protocol authenticate the user. Afterwards the encrypted commands and data are exchanged over TCP.

1.2 | ML in Cyber Security

Detecting anomalies, chiefly cyber-attacks, in large volumes of streaming data in any client-server setting is a key problem for cybersecurity, which becomes increasingly significant as the quantity and variety of data collected from various devices continues to expand. This is especially true for IoT and Industrial IoT (IIoT) systems, often embedded in pivotal communication networks. Therefore, the necessity to find valid and universally applicable methods for predicting cyber-attacks combined with the impossibility to use real-world data due to privacy issues, has led to the development of a wide variety of artificial datasets: *e.g.* Edge-IIOT-2022 ([6]), X-IIoTID ([1]), and MQTTset ([14]). In our paper, owing to the hardware and software limitations to which we are constrained, we do not aim to implement a full-fledged simulation of an IoT system, but we do attempt to realistically simulate network traffic, extracting the most relevant information to create a valid Intrusion Detection System.

1.3 | Plan of the Paper

In Section 2, in order to gain insight into the most relevant features of network traffic and cyber-attacks, we focus on the analysis of previously existing datasets, mainly the Edge-IIoT 2022 dataset, reportedly one of the milestones in the field, [10]. Then in Section 3, we create a machine-independent dataset to train a decision tree that will distinguish normal traffic between clients and a server from traffic due to a Denial of Service or DOS attack on the server. In Section 4, we carry out a similar investigation for the case of the "brute force" password guessing cyber-attack. In Section 5, we explore the trade-off between accuracy and privacy by employing the ϵ-differential privacy technique. Conclusions and directions for future work appear in Section 6. In the Appendix, we describe some theoretical aspetcs of the tool we used, in particular ML algorithms and evaluation metrics. All the Python code which we have written is available in our GitHub repository, [9].

2 | Getting to know the Edge-IIot 2022

The Edge-IIoT 2022 ([6]) is an artificial dataset created ad-hoc for the purpose of evaluating IDS solutions tailored for IoT/IIoT systems. The data it contains is comprehensive and realistic: it is composed of 61 highly correlated features with approximately 150 000 observations in a realistically unbalanced proportion of normal (85%) and attack (15%) traffic, collected over a non-continuous time period from November 2021 to January 2022. The dataset includes 14 different types of cyber-attacks, and is thus suitable for both binary (normal vs anomalous traffic) and multi-class (distinguishing between attack types) classification. As reported in [10], the data allow for the creation of a highly accurate binary-class IDS (with supervised ML approaches), but the training of a classifier for heterogenous attack types is a less trivial matter owing to the relatively small number of datapoints for some classes. Numerous ML approaches have already been tested and compared on the dataset, including decision trees and gradient boosting ([3]), ensemble learning ([7]), and deep learning ([8]). In this paper, we resort to the supervised algorithm with the highest reported performance for binary classification, *i.e.* decision trees ([10], see Appendix A) and compare it to the performance of two unsupervised algorithms, K-means clustering and DBSCAN (see Appendix B), on the unlabeled dataset. Other algorithms such as Adaptive Boosting were not considered due to their relatively poor performance in the IDS problem when compared to decision trees, [10]. This comparison between supervised and unsupervised methods is carried out with the purpose of assessing the extent to which the training of supervised ML methods on labelled datasets can improve cyber-attacks' detection as opposed to models with no previous information.

2.1 | Exploratory Data Analysis and Data Processing

As observed by going through the main variables of he Edge-IIot 2022, the latter is a rather complex dataset that contains a total of 64 variables including all the TCP features discussed in section 1 along with their corresponding statistics for other protocols like http and mqtt.

To prepare the Edge-IIOT-2022 dataset for ML analysis, we first removed 18 non-numeric/non-categorical features. We also removed the 'attack_type' feature, indicating the different types of attack, and trained the models using just the binary 'attack_label' column in pursuit of greater accuracy and simpler models. The following steps have been implemented in Python using the scikit library.
Furthermore, for unsupervised algorithms, we performed an additional preprocessing step by standardizing the data (ensuring it has zero mean and unit variance). Thus, if μ_j and σ_j denote respectively the mean and std. deviation of feature j, the transformed data point (for row i) is given by

$$x_{ij-standardized} = \frac{(x_{ij-original} - \mu_j)}{\sigma_j}$$

for observation i. This transformation is particularly important for clustering algorithms (see Appendix B), because the latter rely on distance-based metrics to assign data points to different clusters. In fact, if the raw values of some features are larger in absolute value or are more spread relative to their mean, they can have a disproportionate effect on the decision boundary of the model. This step was unnecessary for the decision tree because it is a non-parametric model based on partitioning the feature space through "if" conditions and is thus independent of the scale of variables.
In an attempt to train algorithms that would be able to generalize well to new data, we removed also some machine-dependent features collected in the dataset, namely the identification numbers of the ports used by the various protocols.

2.2 | Decision Tree Outcomes and Feature Selection on Edge-IIot 2022

As common practice in ML, we divided the data into 80% for training and 20% for testing by setting an arbitrary random seed for the splitting, in order to ensure the replicability of our results. We limited the maximum depth of the tree to 20, in light of our aim of obtaining a simple model, and evaluated its performance using the classification report function, which showcased the following results, where 0 denotes the class of cyber-attacks while 1 denotes the class of normal traffic (for the definitions of the evaluation metrics see Appendix C):

	precision	recall	f1-score	support
0	0.97	0.86	0.91	4985
1	0.97	1.00	0.98	26575
accuracy			0.97	31560
macro avg	0.97	0.93	0.95	31560
weighted avg	0.97	0.97	0.97	31560

The high accuracy of the model shows that the decision tree was indeed able to distinguish quite well between cyber-attacks and normal traffic but underestimated the number of connections caused by an attack, as can be seen from the relatively lower recall of class 0. To verify whether this relatively simple model could be improved, we tried to tune the hyperparameters of the decision tree. Using a cross-validated random search through a list of possible parameter values, different models were trained on the training part of the dataset to determine which of them maximized the accuracy metric, using the pre-implemented function RandomSearchCV. The optimal values identified with this procedure as well as the structure of the best decision tree are presented below:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, max_features=1.0, min_samples_leaf=4,
                       min_samples_split=10, random_state=5)
```

Figure 2.1: DT with optimal parameters

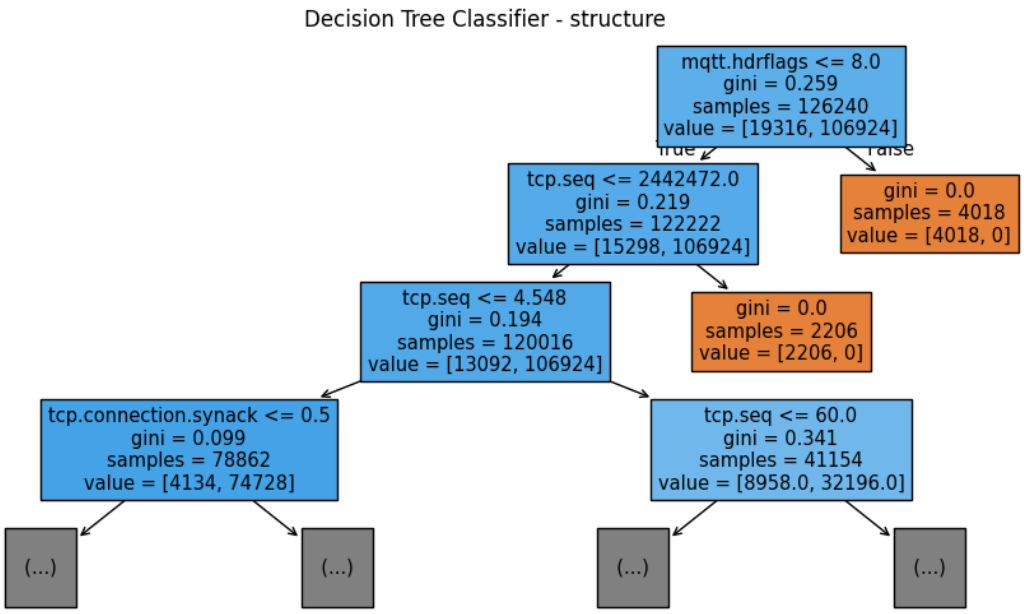


Figure 2.2: Structure of optimal DT

The average cross-validated[1] accuracy of this decision tree remains almost unchanged compared to the original at 0.9745 with std. deviation 0.002 and the classification report shows no improvement in the recall rate of the class of cyber-attacks. Still, the results obtained from either decision tree are more than satisfactory and illustrate the performance that can be achieved with ML methods in cyber security. As can be observed from the figures above, the best decision tree obtained is even simpler than the one first

[1]In repeated K fold, the dataset was divided into five folds. For five times, a new fold is reserved for testing and the remaining four folds are used as data for testing. After each of the folds has been used for testing once, the data is shuffled so that the folds have new configurations, and the same five-step process is repeated. This continues for five rounds for a total of 25 cycles of training and testing. The advantage of using repeated k fold for cross validation is that it shows how well (or poorly) the model generalizes to unseen data. It also reduces the risk of overfitting to a single train-test split.

obtained and yet it performs incredibly well on the dataset. This shows that a too complex model might run into the risk of overfitting and not generalising well to unseen data in this context. Therefore, we further analysed the importance of the various features according to the decision tree, to understand the origin of its predictive success. It was found that most of the available columns were not considered in the data partitions performed by the decision tree and instead only 10-15 features were used (as clear from the figure below). Therefore, it is possible to reduce the high dimensionality of the dataset by removing all those variables unnecessary for predictions and focusing on the analysis only of tcp.seq, mqtt.hdrflags, tcp.ack, tcp.connection.synack, tcp.len, tcp.flags, ... These findings were indeed found to be correct since a decision tree trained only with this subset of features performed equivalently to the model with all covariates in terms of accuracy and recall (of the attack class).

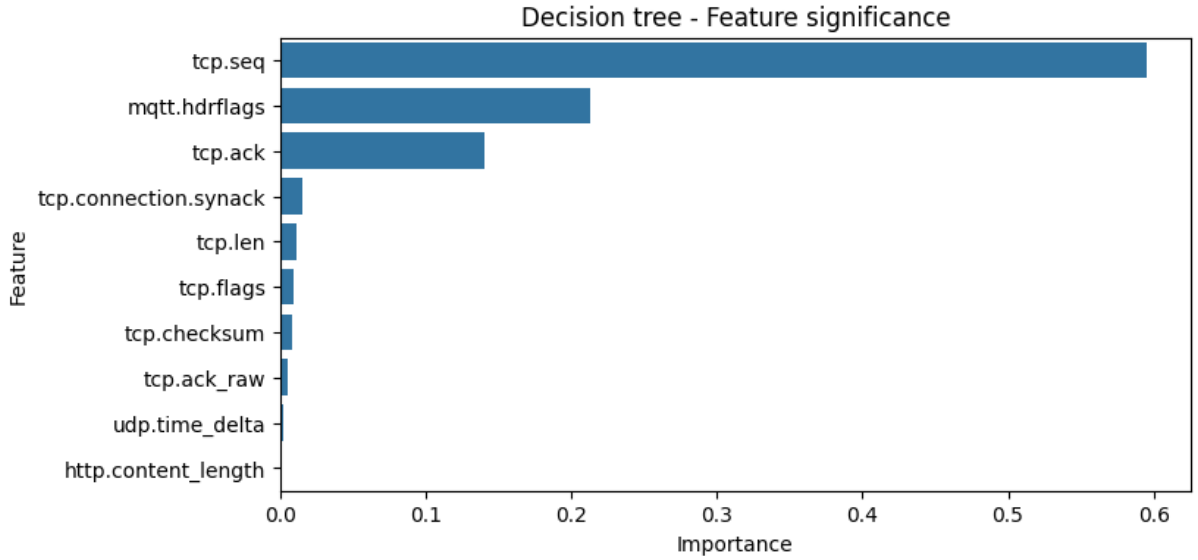


Figure 2.3: Feature importances according to optimal DT

2.3 | Clustering and PCA for cluster visualization on Edge-Iiot 2022

We then compared the decision tree's performance to the two unsupervised algorithms, k-means and DBSCAN, that were used to group data points on the whole dataset as either belonging to attack or normal traffic. Therefore, we fitted the K-means algorithm on the standardized dataset to split it into two clusters and compared such labels with the correct classification of each observation. Specifically, since K-means (and also DBSCAN) only detect different groups of data points with similar features, thus assigning them arbitrary labels, we had to exchange the predicted labels (i.e. normal traffic or attack type) within each cluster so that they matched the highest occurring true label in that cluster.
The classification report shows that both the accuracy and the recall of class 0 (as well as the other metrics) were significantly lower compared to results obtained for the decision tree. In the case of DBSCAN, as explained also above, the algorithm searches for an optimal number of clusters given some input parameters that dictate clusters' properties. Therefore, we tried to implement different variations of DBSCAN for various combinations of parameter values, namely min_samples_split ranging from 1000 to 5000 in steps of 500 and epsilon ranging from 0.1 to 1 in steps of 0.2. While the number of clusters selected by the algorithm usually exceeded 2, there was a unique case for min_samples_split = 1500 and epsilon \approx 0.3 in which two clusters were detected. We resorted to this instance to compute the metrics relative to our binary classification problem of normal vs anomalous traffic. It was found also in this case that the clustering algorithm was not suited to differentiate between the two classes:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	24301
1	0.83	0.88	0.86	133499
accuracy			0.75	157800
macro avg	0.41	0.44	0.43	157800
weighted avg	0.70	0.75	0.72	157800

Figure 2.4: Classification report of K-means

	precision	recall	f1-score	support
0	0.02	0.00	0.00	24301
1	0.84	0.97	0.90	133499
accuracy			0.82	157800
macro avg	0.43	0.49	0.45	157800
weighted avg	0.72	0.82	0.76	157800

Figure 2.5: Classification report of DBSCAN

To better visualize the clusters and evaluate qualitatively how much they differed from the true ones, we performed PCA on the dataset, thus plotting the two directions of the dataset's maximum variance, colour-coding the data points for the cluster being considered (blue denotes normal traffic and yellow instead anomalous traffic):

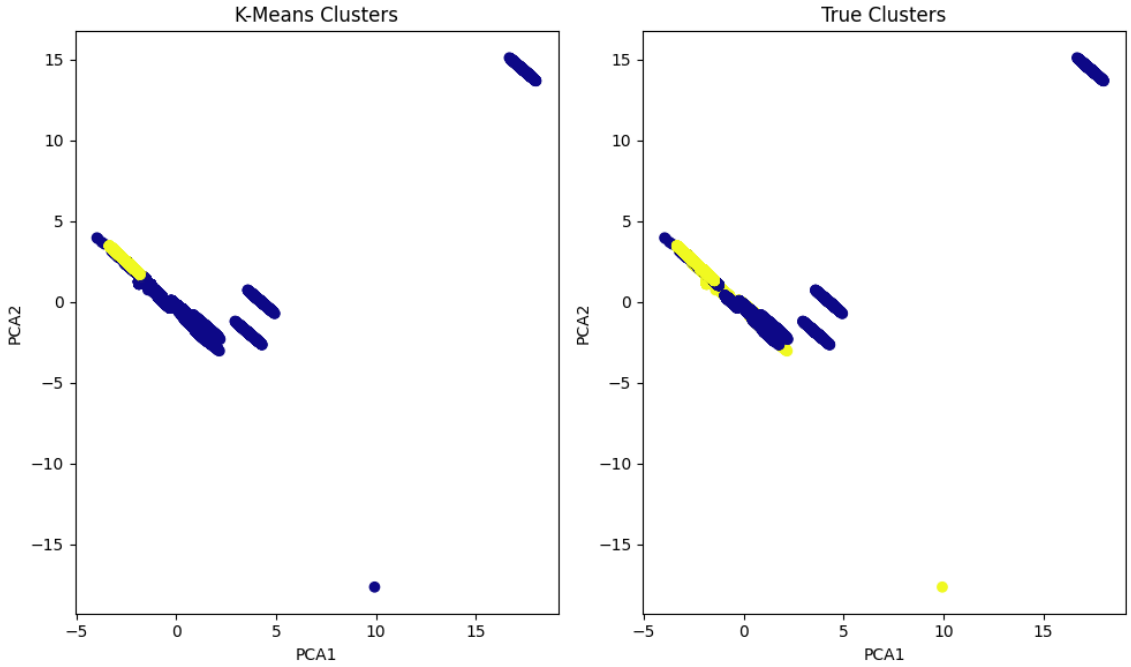


Figure 2.6: Datapoints colour-coded for K-means against true clusters

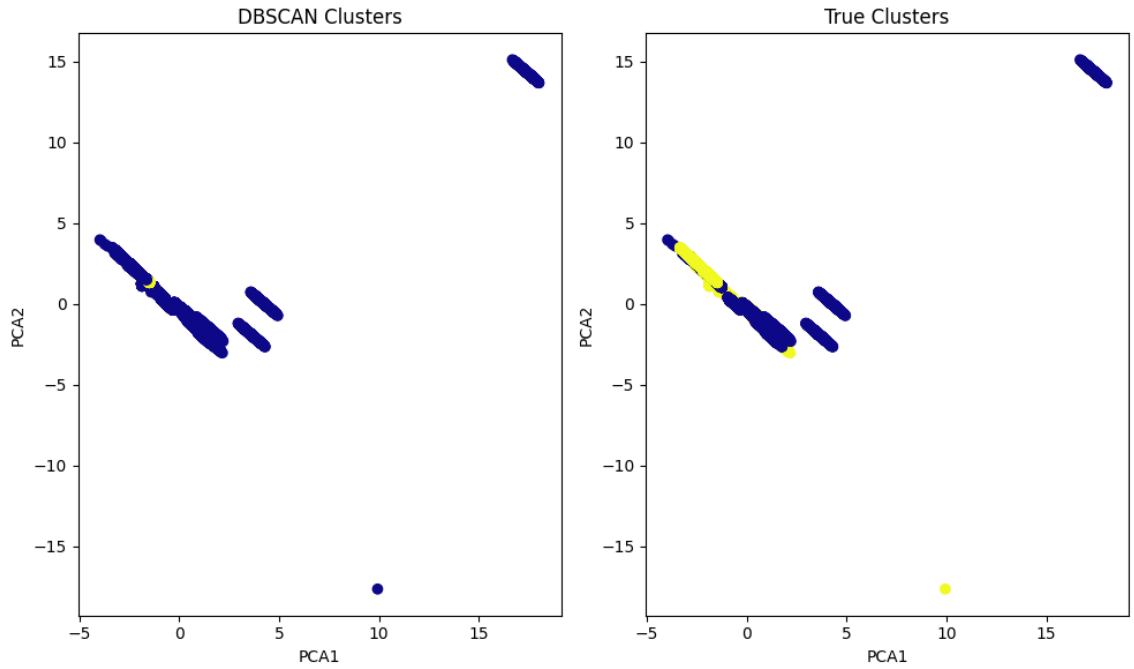


Figure 2.7: Datapoints colour-coded for DBSCAN against true clusters

Overall, these clustering algorithms show both qualitatively and quantitatively a much lower performance compared to decision trees with recall rates of the attack class almost identical to zero. In addition, it is significantly more difficult to interpret the common characteristics of data points within a given cluster and thus also to understand which features are truly important for predictions. This shows that unsupervised algorithms applied directly to the data are not suited for the binary classification problem of distinguishing normal and anomalous traffic in this cyber security setting. This justifies the creation of custom labelled datasets to train suitable ML algorithms.

3 | Our DOS Attack Dataset

In this section, we will attempt to create a realistic and machine-independent dataset to train a decision tree that will distinguish normal traffic between clients and a server from traffic due to a Denial of Service or DOS attack on the server.

3.1 | Simulations and Dataset Creation

First, we focused on generating realistic normal traffic between a given number of clients and a server, which we implemented on our local devices using the Python socket library. Inspired by the site [12], we created two Python scripts to simulate the exchange of packets between multiple clients and a multi-connection echo-server, which would send back the same messages it received. Since both clients and servers were locally implemented on our devices, it was not possible to make them exchange information using http packages about the given sites' information, as would be the case for a web server. Instead, the clients and server sent and received respectively simple TCP packets with relatively short string messages encoded using utf-8. The length of the strings, and thus of the TCP messages, was randomly selected from a Gaussian distribution with a mean of 100 characters and variance of 20. Moreover, also the content of the string messages was randomly generated. Furthermore, the number of clients accessing the server was kept constant throughout the simulation. Even if this might not appear to be realistic, at first, we realized that the traffic generated by a new client was equivalent to a higher quantity of messages sent by the already existing clients. As a matter of fact, in the final dataset, these two phenomena were completely equivalent as the port numbers of the clients were to be removed since they represented machine-dependent features, not relevant for a general analysis of DOS attacks. Therefore, despite the constant number of clients, the quantity of network traffic was modelled in such a way that it could vary significantly over the time period of the simulation. In fact, by updating the time delay between consecutive messages based on the previous pauses, it was possible to simulate "peak" periods of high network traffic and periods in which just few messages were exchanged. The delays of the messages sent and received thus evolved like in a dynamical system with 2 main phenomena:

- Standard background traffic periods, in which the value of delay undergoes small random fluctuations, which are normally distributed and constant over time, and experience larger sinusoidal and time-dependent fluctuations, which mirror the natural evolution of traffic on real servers over longer time periods.
- Peak traffic periods, in which once the time delay between messages drops below a given threshold (and so the client-server traffic has already increased substantially), there is a much faster polynomial-time growth of traffic. After gradually reaching a maximum amount of traffic, the number of messages exchanged between the server and clients slowly decays in polynomial-time back to the original amount of standard background traffic. In this way, we were able to simulate the evolution of network traffic on a hypothetical web-server in which there are sometimes sudden and substantial changes in traffic as a result of individuals sharing some news, or some other social effect, which features a breakout point.

The next step was to find a way to simulate the DOS attack on the local IP address of the server. To execute the attack, we resorted to the Python script created by [13], thanks to which we were able to simulate DOS for an arbitrary time interval with an adjustable number of threads (affects how much anomalous traffic is generated).
Finally, we were ready to run the full simulation with both normal and anomalous network traffic, collecting all the relevant features using the Wireshark application, [15]. Therefore, we run two simulations lasting approximately 2.5 hours in total with two DOS attacks with a duration of 30 seconds. This was done to make sure that the proportion of observations corresponding to anomalous traffic were realistically much lower than the number of rows for normal traffic, aiming ideally for 15% of attack traffic as in the Edge-IIoT 2022 dataset.
For each simulation, we initialized a server on the IP address for loopback traffic capture "127.0.0.1", which received packets on an arbitrary port number (in our case 12345) higher than 1000 (lower port numbers might be used by operating system and so unavailable for usage). Then, once the server was listening on the given port, the Python script for the clients was run and the TCP packets began to be transferred between clients and servers. Then, the script for the DOS attack was run for a duration of 30 seconds and using only 50 threads. This choice was made due to some technical difficulties with the duration of the simulation, and also in an attempt to align the promotion attack traffic to the dataset

above. Using Wireshark, we first analyzed the density of traffic in given time intervals to evaluate, at least qualitatively, the differences between normal and anomalous traffic and to validate the simulated data's realism. Specifically, we relied on the following two graphs (with logarithmic scale) of normal and anomalous TCP packets recorded every 5 seconds:

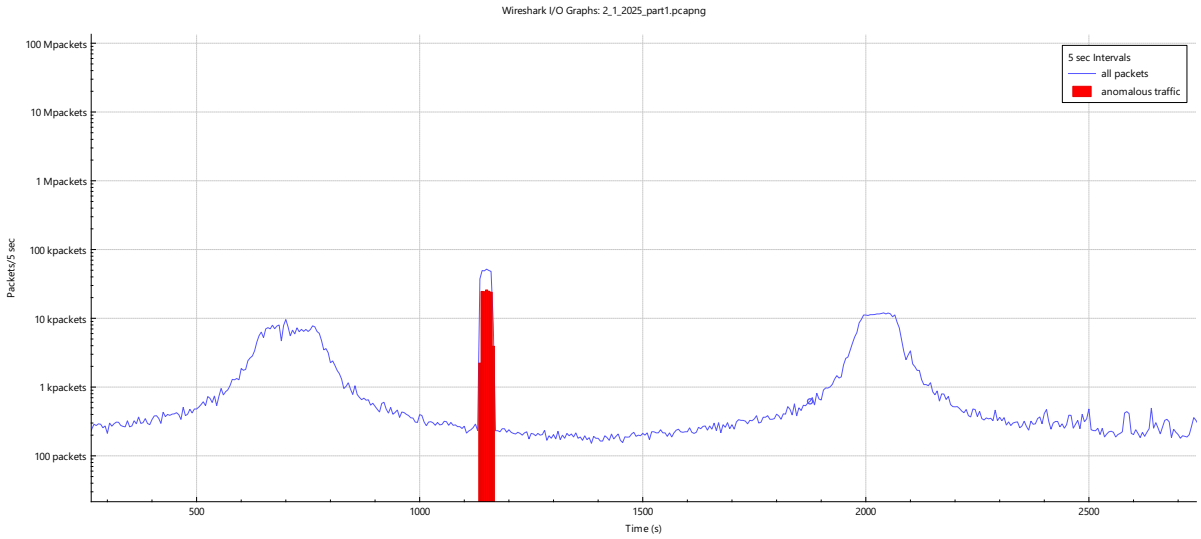


Figure 3.1: First simulation of normal traffic with DOS attack

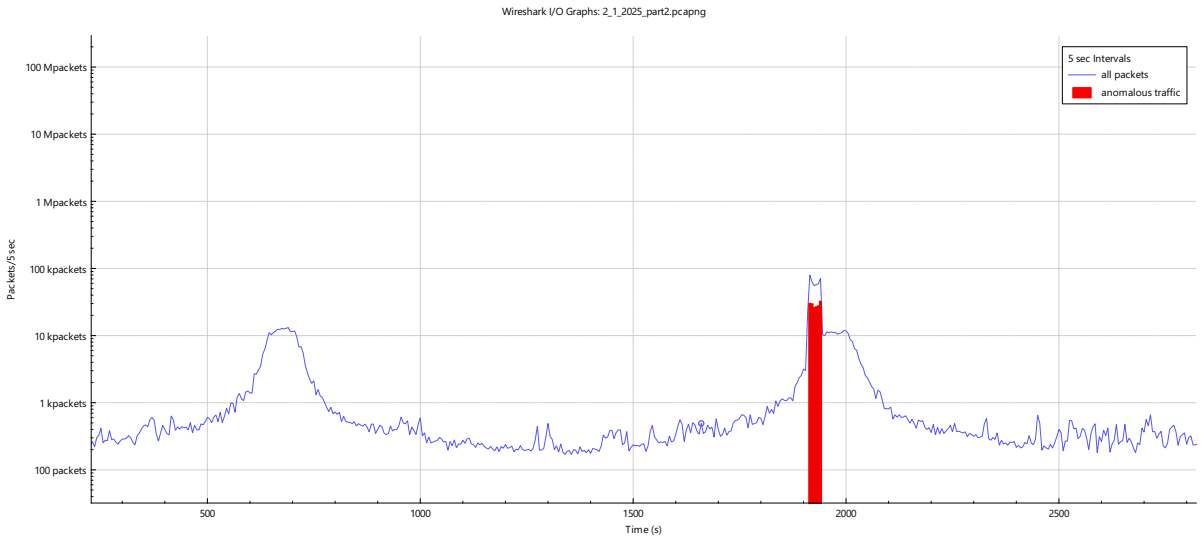


Figure 3.2: Second simulation of normal traffic with DOS attack

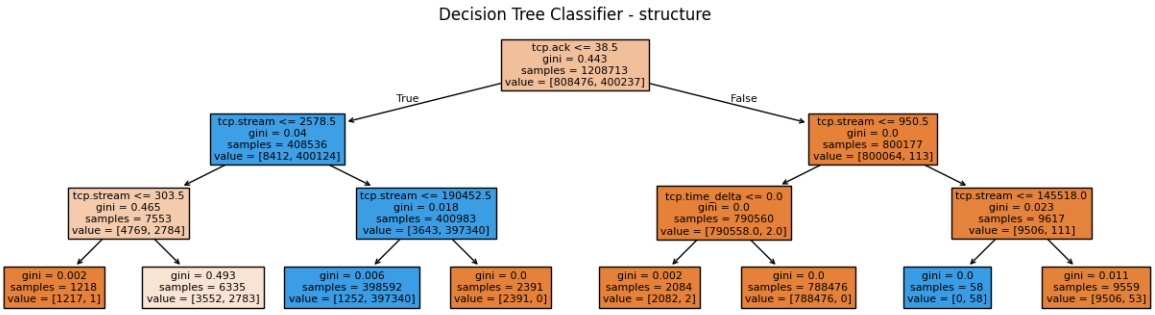
The graphs revealed significant differences, with normal traffic showing a relatively steady rate punctuated by naturally occurring peaks, while the DOS attack periods exhibited sharp and sustained spikes in traffic density. These plots are thus crucial not just in understanding the traffic patterns, but also in validating the simulated data's realism.

3.2 | Dataset's Analysis with DT

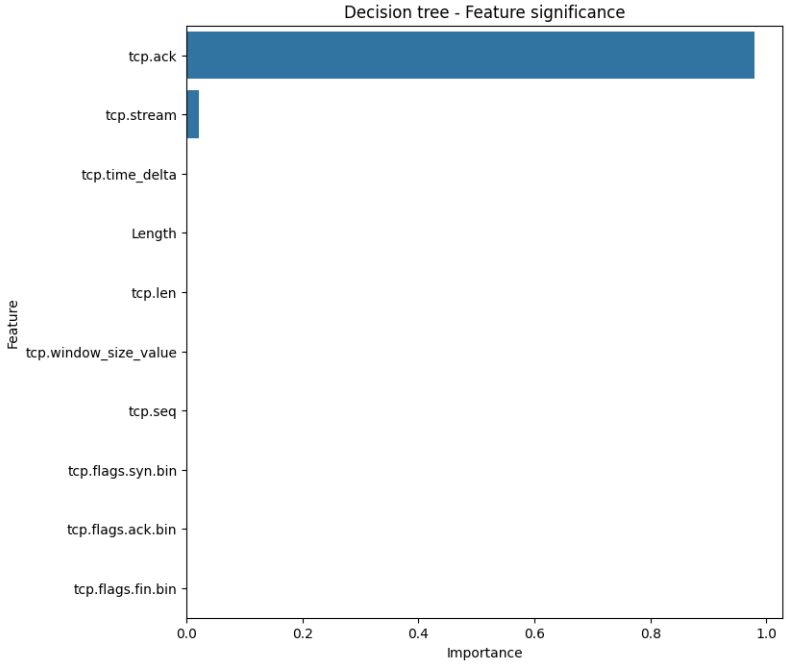
Once the two simulations were completed, the data, with the TCP columns found relevant in the Edge-IIot 2022 dataset, was exported from Wireshark into a csv file. After a few data cleaning steps, which included removing any incomplete entries, handling any missing values and removing any residual traffic from sources outside the simulation, the two datasets were concatenated into a single data frame. The non-numeric column entries were then removed, and each observation of the dataset was then labelled as normal (class 0) or anomalous (class 1). To perform this task, we selected all messages in the time

intervals in which we had run the DOS attack script and labelled as anomalous those that had a port number different from the server port, 12345. We found that the percentage of anomalous traffic amounted to 33% of the all observations and despite being significantly higher than the one in Edge-IIoT 2022, it is still realistic enough. We then proceeded to train a decision tree on 60% of the dataset's observations setting its maximum depth to 3 in order to obtain a simple and interpretable model. The classification report for this model, despite its simplicity, shows incredibly high results:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	539396
1	1.00	0.99	1.00	268515
accuracy			1.00	807911
macro avg	1.00	1.00	1.00	807911
weighted avg	1.00	1.00	1.00	807911



We then moved on to analyze which features were most important for predictions and since the most important trademark of the DOS attack is the incomplete 3-way handshake, leaving the server waiting for additional info, it is not surprising that the most important feature is indeed tcp.ack.



3.3 | Dataset's Validation with K-means

In this subsection, we run the K-means unsupervised algorithm on the unlabeled version of the dataset in order to evaluate whether our dataset was too simple/unrealistic compared to the Edge IIoT 2022, thus

testing the validity of the above performance metrics for the decision tree. By running the K-means with two clusters on the whole dataset we obtained the following classification report. On the unlabeled version of the dataset, we also run the K-means clustering and use it as a benchmark to evaluate whether our dataset was too easy to be analyzed or was instead of comparable difficulty compared to the Edge-IIoT 2022. The two groups of data points found by the algorithm are presented below:

	precision	recall	f1-score	support
0	0.80	0.99	0.89	1347170
1	0.98	0.50	0.66	667353
accuracy			0.83	2014523
macro avg	0.89	0.75	0.77	2014523
weighted avg	0.86	0.83	0.81	2014523

The recall of the attack class (category 1) shows that the clustering algorithm mislabeled attacks as normal traffic for half of all anomalous traffic observations. Moreover, also the overall accuracy of the model is significantly lower compared to the values attained by the decision tree. In addition, despite being higher than the measurements obtained for the clusters created by K-means in the Edge IIot 2022, both statistics are still of comparable magnitude. This confutes the previous hypothesis that our dataset was too simplistic for the ML algorithms employed and thus supports our conclusions concerning the decision tree in the previous section. The recall of the attack class (category 1) indicates that the clustering algorithm incorrectly labeled attacks as normal traffic for half of all anomalous traffic observations, thus exposing its great shortcomings. Additionally, the overall accuracy of the model is significantly lower compared to the values achieved by the decision tree in section 3.2. Despite these metrics being higher than those obtained for the clusters created by K-means in the Edge IIoT 2022 dataset, the latter are still of comparable magnitude. This refutes the earlier hypothesis that our dataset was too simplistic for the ML algorithms used, thereby supporting our conclusions regarding the decision tree in the previous section. To better grasp the evaluation metrics reported above, we performed principal component analysis (PCA) on the dataset in order to plot K-means clusters against the true clusters. In this way, we managed to verify our findings also qualitatively (blue data points represent normal traffic and yellow data points represent anomalous traffic):

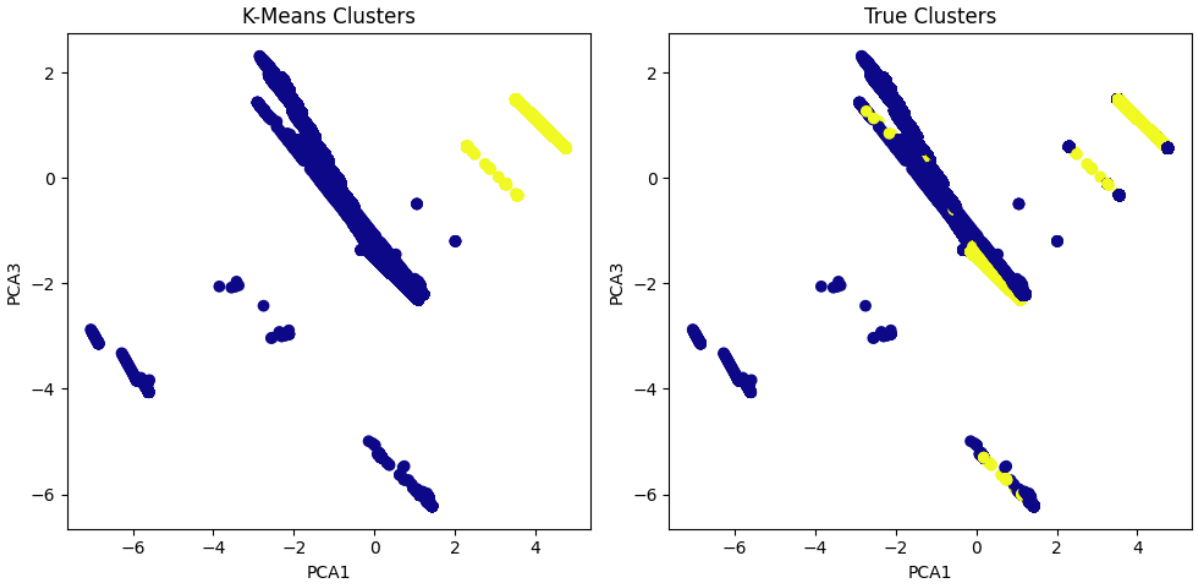


Figure 3.3: Visual comparison of K-means and true clusters

4 | Our Brute Force Attack Dataset

As a second simulation of cyber-attack we choose the "brute force" password guessing. We opted for this specific instance because it is a common and practical method used by attackers to exploit weak authentication mechanisms.
This simulation is based on the SSH protocol, a brief introduction to it appears in Section 1.1.

4.1 | Simulations and Dataset Creation

In order to carry out the simulation we set up an online server with the following features (using the Linode cloud hosting provider): Ubuntu 24.04 LTS, 1 CPU Core, 25 GB Storage, and 1 GB RAM. We set up 2 users on this remote server with the respective passwords.
For the normal traffic we created a script called "normal_SSH.py" which can be found in the github repository. We repeatedly ran 6 different simple Linux commands intended for gathering system information and testing SSH traffic. Each time we logged in with one user by entering the correct password and after executing the command we logged out. The commands we executed are the following:

- ls -l : lists files and directories in the current directory. It shows details such as permission, ownership, size, and modification date
- df -h : prints disk space usage
- whoami : tells the logged-in users's name
- cat /etc/os-release : prints the system information such as name, version, and other details
- uptime : shows how long the system has been running, number of users, and the system load averages
- echo 'Testing normal SHH traffic' : it simply prints the string.

The main Python library used for establish SHH connections to remote servers and implement SSH protocol is "paramiko", [11].
On the other hand, in order to simulate the brute force attack we used a script that attempts multiple username-password combinations to connect to a remote server. Indeed in the repository there is also a CSV file named passwords.csv which is used in the attack simulation. In order to avoid legal and ethical issues we run the malicious script on a Kali Linux virtual machine. This particular machine is specifically designed for security researchers. In addition to acting as a sandbox for our main machine, it includes a vast collection of tools used for testing security. The specifications of the VM are Linux (64-bit, kernel 5.x), 2.5 GB RAM, and 1 vCPU. So we ran first the script simulating normal traffic for about 5500 seconds (about 80 minutes) and around the 1000th second (16th minute) we executed the malicious script which lasted approximately 100 seconds, the time it took to attempt the login with 500 different passwords. As we can see from the graph below, there is a significant increase of exchanged packets around the period mentioned above.

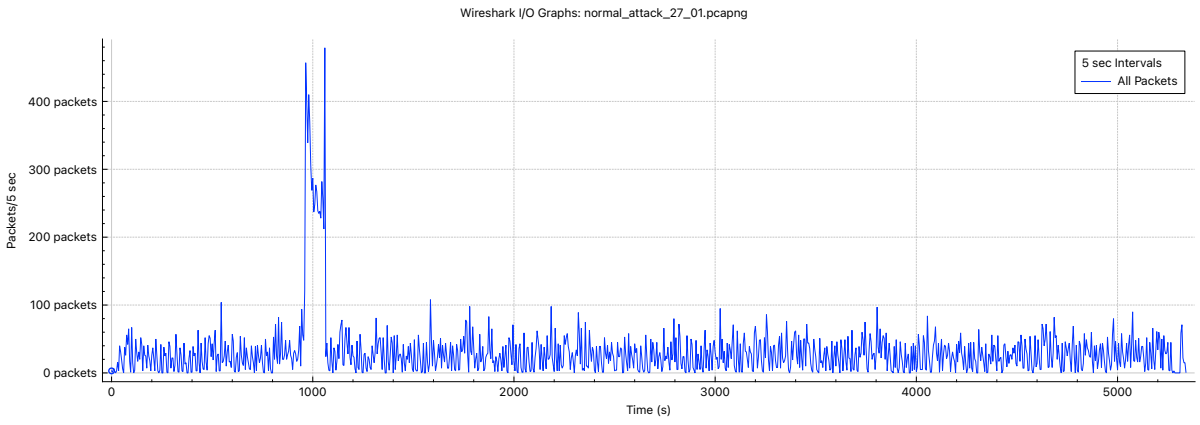


Figure 4.1: Wireshark graph of traffic for Brute Force simulation

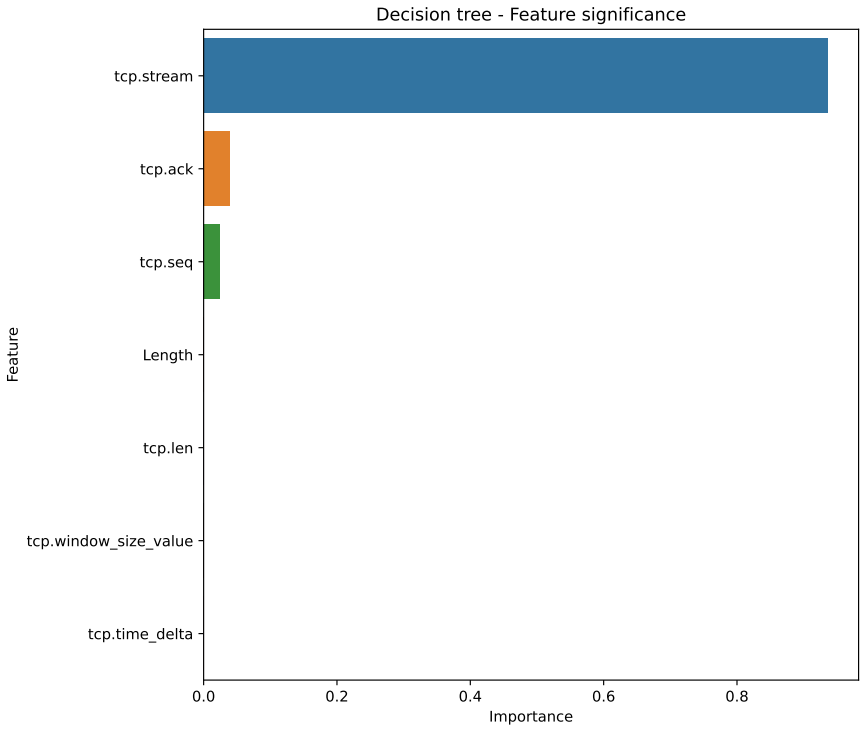
With Wireshark we have been able to capture the traffic during the simulation and especially we registered the TCP and SSHv2 (newer and more secure version of SSH) packets exchanged between the online server and the virtual machine. First of all we notice that both during the normal and the attack part there is a sequence of alternating TCP and SSH packets with almost the same proportion over the whole period of simulation. This was as expected, since the SSH protocol is based on the TCP one. The main features we decided to record for the anomaly detection are mainly the ones reported in section 1 of the introduction and they are aligned with those found to be most important also in the previous sections.

4.2 | Dataset's Analysis with DT

As we did for the DOS attack we proceed with the anomaly detection analysis using the decision tree. First of all we encoded all the binomial categorical variables in 0 or 1's (e.g. "set" $\rightarrow$ 1, "not set" $\rightarrow$ 0). Then the non-numeric column entries were removed. So we took care of the labeling process to ensure the data is categorized for supervised learning. For this task we used the fact that we knew the exact interval time at which the attack was run. The result is that we got around 22% of attack traffic, which is more aligned with the percentage of the Edge-IIoT 2022 dataset (15%). We then proceeded to train a decision tree on 60% of the dataset's observations setting its max depth to 4 in order to obtain a simple and interpretable model. The performance metrics for this model, despite its straightforward design, demonstrate very high outcomes, but still a bit worse than the ones found in Section 3.2.

	precision	recall	f1-score	support
0	1.00	0.99	1.00	7869
1	0.98	1.00	0.99	2239
accuracy			0.99	10108
macro avg	0.99	1.00	0.99	10108
weighted avg	0.99	0.99	0.99	10108

We then moved on to analyze which features were most important for predictions:



As we can see tcp.stream has the highest importance (close to 1.0). The most plausible explanation of this result is that, during the attack period, multiple SSH connections are established (and so also the relative TCP connections) in a rapid succession and so the value of tcp.stream is changing more frequently than

in the normal period. Indeed in the normal SSH simulation, the user is staying connected to the server for a longer period and so also the TCP session persists longer.

4.3 | Dataset's Analysis with k-means

We proceed our analysis by running the k-means algorithms on the unlabeled dataset. Again our goal is to understand whether our dataset is too simple/unrealistic. By running the K-means with two clusters on the whole dataset we obtained the following classification report:

	precision	recall	f1-score	support
0	0.82	0.97	0.89	19610
1	0.71	0.26	0.38	5658
accuracy			0.81	25268
macro avg	0.77	0.62	0.64	25268
weighted avg	0.80	0.81	0.78	25268

The recall value for the attack class (1) (26%) highlights a bad ability in classifying the elements of this class, there are too many False Negatives. On the other hand, at least our model does not mistakenly classify data as an 'attack' when they actually belong to the simulated normal traffic. To grasp also qualitatively the performance of the k-means we first perform PCA and then plot K-means clusters against true clusters. As we can see from the plots we can see that the k-means was able to identify the general clustering structure in the data, but its assignments are not perfect. Misclassifications are evident in regions where the clusters are closely spaced or overlapping, which suggests that this algorithm struggled with ambiguous points near the cluster borders. In the end we can say that k-means is less effective in detecting brute-force rather than Dos attack (the recall of the attack class is indeed lower), but again it is confirmed that k-means is not well suited for this task.

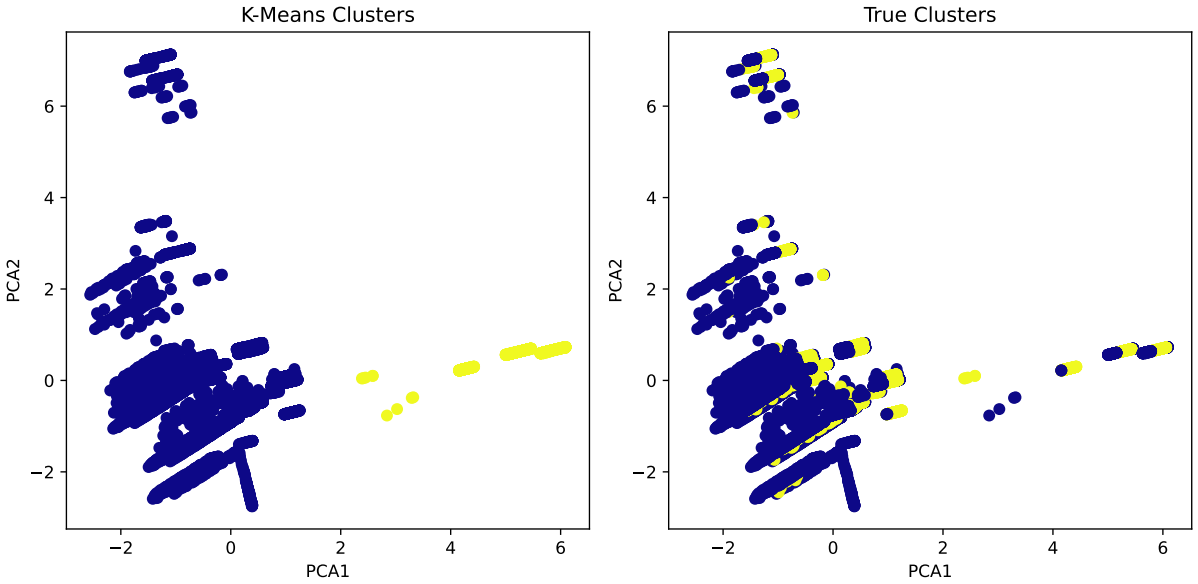


Figure 4.2: Comparison of K-means and true clusters

5 | Differential privacy

While the primary objective of anomaly detection is to yield high detection accuracy, the requirement of privacy is also paramount. The latter requires that no sensitive information is leaked to untrusted parties. One of the most successful standards for asserting formal privacy guarantees is differential privacy (DP). Assuming that the specifications of our network recordings are in a sense vulnerable data and need to be protected from unauthorized access, in this section we investigate how a differential privacy mechanism can be added to our ML model, in order to guarantee the privacy of data, and we analyze how performance is affected.

5.1 | Introduction

Protecting data privacy has been a major concern in many applications, because sensitive data are being collected and analyzed. In the early 2000's, differential privacy ([4, 5]) has been proposed as a criterion to guarantee the privacy of input data from the output. More precisely, DP ensures that by looking at the statistical results calculated from a dataset, one cannot tell (in probabilistic terms) whether the input data contain a certain record or not. More formally, an (ϵ, δ)-differential privacy mechanism $\mathcal{M}$ is a randomized function that maps input datasets D to outputs $\mathcal{R}$ such that the following property is satisfied for any two adiacent datasets D and D' (differing by at most one record) and for any subset $S \subseteq \mathcal{R}$:

$$Pr[\mathcal{M}(D) \in S] \leq e^{\epsilon} Pr[\mathcal{M}(D') \in S] + \delta \ ,$$

where

- $\epsilon \geq 0$ (privacy budget) controls the privacy loss (smaller ϵ means stronger privacy guarantees);
- $0 \leq \delta \leq 1$, when δ is different from 0, the original ϵ-DP is relaxed to approximate differential privacy, (ϵ, δ)-DP.

In our case, we focus on the original (strict) ϵ-DP definition (*i.e.* we take $\delta = 0$).
The most common method for achieving DP is injecting calibrated noise into various stages of the process, including the input data itself, the output of ML models, or even the model weights or internal parameters. This can be done using the "Laplacian mechanism", which works by adding noise coming from the Laplacian distribution[2] to each feature of the dataset. Namely, once the privacy budget ϵ is chosen, for each feature A of the dataset, the scale parameter of the Laplacian is usually taken to be the "l_1-sensitivity" of the dataset over the privacy budget ϵ, where the l_1-sensitivity amounts to the maximum absolute difference between any two possible values in the feature A:

$$\text{lap}(A) = \frac{l_1\text{-sensitivity(A)}}{\epsilon} \ ,$$

where l_1-sensitivity(A) $= max(A) - min(A)$.

5.2 | Our approach

We aim to investigate whether the employment of differential privacy in our framework leads to a significant reduction in model accuracy or if the performance remains satisfactory. Our setting is the one of Section 3, *i.e.* anomaly detection of the DOS attack through the Decision Tree algorithm. In particular we choose to inject Laplacian noise on the row data after they are split in "train" and "test" for the Decision Tree. To be specific, we add the noise to all the features apart from the flag-features and the discrete features, since the Laplacian noise is obviously a float number. Again, the complete code of our experiments can be found at the github page, [9]. As we can see from Figure 5.1 below, we decided to test different values of the privacy budget ϵ, namely all the integers from 1 to 10. Such relative high budgets is justified by the fact that our investigation deals with a machine learning algorithm (Decision Tree). Indeed in ML the value of ϵ is taken slightly bigger than the one normally used in the database query setting (see *e.g.* [2] for more details). Simultaneously, we also decided to compare the performance of Decision Tree algorithm while varying the hyperparameter "max depth". This is crucial for optimizing the trade-off between model complexity, performance, and privacy. As expected, as the privacy budget decreases we have a decrease

[2]The probability density function of Laplacian distribution is: $f(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$, where μ is a location parameter and $b > 0$ a scale parameter.

also in the accuracy of the model. However the level of the performance is still acceptable. An interesting fact is that, regardless of the max depth value, the accuracy tends to converge to 0.83 when the privacy budget is equal to 1.

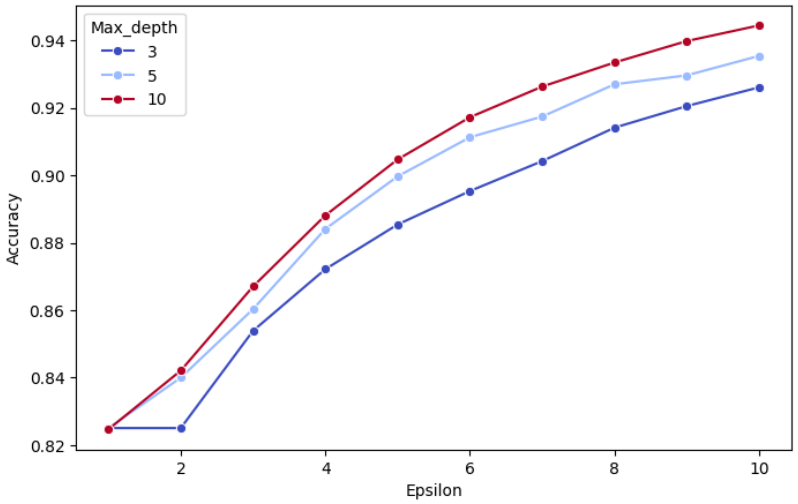


Figure 5.1: Line graph of Accuracy vs Quantity of Noise

6 | Conclusions

In our paper, by conducting a preliminary analysis of an already tested and established dataset, the Edge IIoT 2022, we have derived some important requirements for a good dataset and have enforced the latter in the creation of two artificial datasets for DOS and Brute Force attacks detection. In this process, we have critically analyzed supervised and unsupervised learning algorithms on all these datasets, with the dual aim of evaluating the applicability of ML algorithms in the IDS problem and of validating the complexity of our datasets. To better highlight the differences and similarities between the datasets and the performance of Ml methods, we summarise our findings in the table below: (the recall refers to the recall of the attack class)

	Our DOS Dataset	Our Brute Force Dataset	Edge-IIoT 2022
Decision Tree			
Accuracy	1.00	0.99	0.97
Recall	0.99	1.00	0.86
K-means			
Accuracy	0.83	0.81	0.75
Recall	0.5	0.26	0.00
DBSCAN			
Accuracy			0.82
Recall			0.00

Table 6.1: Performance Metrics for different datasets and ML models

Overall, Table 6.1 demonstrates that supervised ML algorithms, namely decision trees, are suitable to detect a wide variety of Cyber attacks in different types of networks. Differently, unsupervised algorithms, mainly K-means and DBSCAN, showcase a strikingly poor performance because they don't have any previous information about the datasets. Still, such clustering algorithms have the potential to play an important role in comparing the complexity, and so the realism, of different artificial datasets. In fact, by using as a benchmark the performance metrics like accuracy/recall/... of such models with no previous information on the data, one can analyse more critically the advantages brought by supervised algorithms. Accordingly, this further reinforces the necessity to create realistic, labeled and possibly machine-independent datasets to train proper IDS that can effectively detect cyber threats. Yet, the attainment of such an objective in the near future seems to be constrained by a number of factors including hardware/software availability for simulated datasets as well as privacy issues involving data treatment on real networks' data. To address the second problem, we investigated how differential privacy could be implemented on our model for anomaly detection, and we analyzed how it affects the performance. As expected, the higher are the privacy guarantees, the lower is the accuracy of the model, but still the level is good.
Notwithstanding this, we hope that the careful treatment of the topic of ML in cyber security in this paper together with our careful treatment of data in the DOS and the Brute Force datasets, will provide a way to further advance this field of research, by providing easily implementable solutions for any researcher. We still remain conscious of the many limitations of our study and of the many details of both the analysis and the datasets, which can be surely better fine-tuned and will be discussed in the next section.

7 | Future improvements

First, the realism of the simulated datasets created for both DOS and Brute Force could be improved by several means. As a matter of fact, for the DOS dataset, we could try to simulate multiple attacks varying the number of threads used in each as well as the duration, even if this would require a much longer simulation, for the normal traffic to balance the attack traffic. On top of this, as far as the Brute Force dataset, we could experiment different kind of attacks such as Slow brute-force attack (login attempts are spread over an extended period) or distributed brute force attack (attacks originates from multiple IPs). Another idea worth exploring is to perform a multiple anomaly detection mixing the two kinds of attacks. Moreover from the standpoint of the datasets' analysis, due to the potentially very large number of recorded features in a real network, it could be beneficial for both efficiency and performance to first

apply dimensionality reduction to the dataset before training the ML algorithms. In this paper, we have not followed this route since the datasets we used were relatively small, and also because, we wanted to preserve the original features in an attempt to better interpret the results of our models, namely deriving which features are essential for cyberattacks' detection. However, in some situations, it might be beneficial to forego interpretability in favour of simpler datasets as well as computational performance and we must admit that such prospect is quite promising. In fact, when applying PCA to the datasets, we found that in the Edge IIoT 2022 dataset (the one with the most features) more than 90% of the variance of all 40 columns was contained within only 20 principal components and that the last 10 principal components contributed infinitesimally to the variance, as can be noticed from the line-graph below:

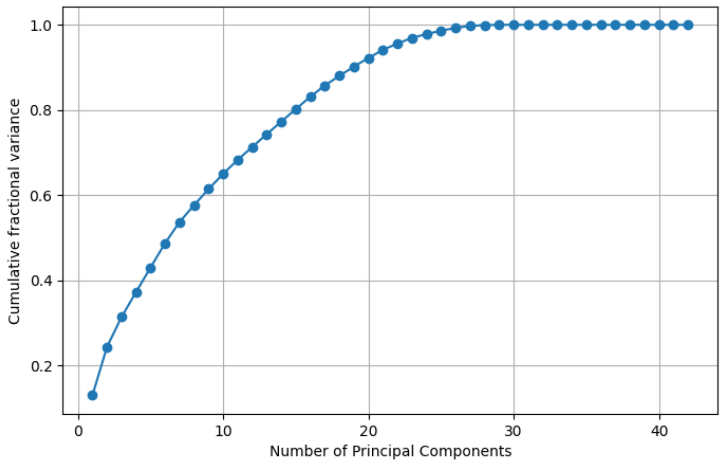


Figure 7.1: Proportion of variance contained in principal components

Finally, as far as the privacy issue, an interesting direction of future investigation is studying other modalities of applying differential privacy. First of all we could investigate how the noise injection works at the different steps of the Decision Tree algorithm. This would require in particular a preliminary deeper analysis of how the Decision Tree algorithm works. Another interesting test would be to use a different sensitivity metrics, such as the l_2-metric, together with noise coming from the Gaussian distribution.
In the broader context of ML, the issue of the trade-off between privacy, performance, and complexity is quite a hot topic nowadays. While there are many works in the literature aiming at combining these apparently contrasting issues, *i.e.* privacy on one side, and performance and complexity on the other side, some authors ([2]) raised serious concerns regarding the possibility of bringing together these issues in the context of ML. An interesting line of future work is then to investigate to what extent the criticisms of [2] could affect our implementation of DP to anomaly detection.

8 | References

[1] M. Al-Hawawreh, E. Sitnikova, and N. Aboutorab. X-iiotid: A connectivity-agnostic and device-agnostic intrusion data set for industrial internet of things. *IEEE Internet of Things Journal*, 9:3962–3977, 2022.

[2] A. Blanco-Justicia, D. Sanchez, J. Domingo-Ferrer, and K. Muralidhar. A critical review on the use (and misuse) of differential privacy in machine learning. *ACM Computing Surveys*, 55:1–26, 2023.

[3] M. Douiba, S. Benkirane, A. Guezzaz, and M. Azrour. An improved anomaly detection model for iot security using decision tree and gradient boosting. *Journal of Supercomputing*, 79:3392–3411, 2023.

[4] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *Theory of Cryptography*, page 265–284. Springer Berlin, 2006.

[5] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, August 2014.

[6] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke. Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning. https://dx.doi.org/10.21227/mbc1-1h68, 2022.

[7] C. Hazman, A. Guezzaz, S. Benkirane, and M. Azrourand. lids-sioel: Intrusion detection framework for iot-based smart environments security using ensemble learning. *Cluster Computing*, 26:4069–4083, 2023.

[8] V. Hnamte and J. Hussain. Dcnnbilstm: An efficient hybrid deep learning-based intrusion detection system. *Telematics and Informatics Reports*, 10:1–13, 2023.

[9] A. Honsell and A. Lamonarca. Python code for anomaly detection. https://github.com/AndreaLamo/Anomaly-Detection-in-Cyber-Security, February 2025.

[10] T. Al Nuaimi, S. Al Zaabi, M. Alyilieli, S. Alblooshi M. Al Maskari, F. Alhabsi, M. F. B. Yusof, and A. Al Badawi. A comparative evaluation of intrusion detection systems on the edge-iiot-2022 dataset. *Intelligent Systems with Applications*, 20:200–298, 2023.

[11] Paramiko. Python library for the implementation of the sshv2 protocol. https://www.paramiko.org/, 2025.

[12] Socket Programming in Python, Guide. https://realpython.com/python-sockets/, December 2024.

[13] Matrix Team. Dos attack python script. https://github.com/MatrixTM/MHDDoS.

[14] I. Vaccari, G. Chiola, M. Aiello, M. Mongelli, and E. Cambiaso. Mqttset, a new dataset for machine learning techniques on mqtt. *Sensors*, 20, 2020.

[15] Wireshark, Version 4.4.3. https://www.wireshark.org/, 2024.

Appendix

A | Decision Trees

Given a matrix of features with which to predict a given class, the decision tree (DT) classifier algorithm will identify comparisons, based on features' values, between data points that will separate as much as possible data under one label from data under another label. In our case, it compares specific characteristics of network traffic to split traffic instances into groups containing either normal or attack labels. It will continue to apply such comparisons, based on simple if conditions, until every group contains only one type of traffic and the node of the DT, for which all contained observations belong to the same class (obtained as a leaf of the DT), is said to be pure.
After training finishes, the nested structure of if-statements partitioning the space of possible feature values can be applied to classify unlabeled traffic instances. This series of true/false comparisons will categorize the new traffic data strictly based on the conditions satisfied by training traffic. Some of the most useful parameters for decision trees, which correspond to the ones adjusted in the hyperparameter tuning of the DT, in Section 2, include:

- max_depth: how many levels the tree can have, starting from the first comparison and adding a new level for subsequent splits of the parent groups.
- max_leaf_nodes: how many split groups of data (leaf nodes) the tree can have. Unlike max_depth, this parameter does not consider the symmetry of the tree, allowing for one group of data to be split for multiple levels while other leaf nodes remain at higher levels without being split.
- min_samples_split: how many datapoints must be included in a parent group for a comparison to be applied.
- min_samples_leaf: how many datapoints must be in each of the leaf nodes resulting from a split in order for a comparison to be applied.
- max_features: how many of the dataset's features can be considered when selecting a feature with which to make a comparison. Sometimes, this parameter can be expressed as a mathematical function, like $\sqrt{x}$ or $\log_2(x)$, of the total number of features.

Furthermore, decision trees are also easily interpetable ML algorithms thanks to their straightforward visualization. In fact, the decision tree's representation offered by python's library scikit-learn permits to visualize the different comparisons carried out by the DT at each level of the latter. Each box includes the binary comparison used to separate the traffic into leaves, the total number of observations that fall into the leaf (samples), the number of observations in each classification formatted as [anomolous, normal] and also the Gini coefficient which is an index of the impurity of a given node (0 for a pure node and close to 1 for a node with a mix of different classes). More formally, Gini is a metric ranging from 0 (the node is pure) to 1 (the node contains a mif of different classes) and defined as:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

where p_i denotes the proportion of observations belogning to class i in that node.
In addition, decision trees also provide a way to assign to each feature its importance, which quantifies with an information theoretical metric the relevance of that column in affecting the model's prediction and loss reduction. More concretely, the feature importance is calculated based on a measure of the improvement in purity from a decision node, in which datapoints of different labels are grouped together, to the leaf nodes, which split the datapoints into more homogenous groups. The overall importance of a feature is thus the sum of the improvement in purity for all decision nodes which use that feature to split the data.

B | Clustering algorithms

Unlike the supervised learning models, the unsupervised models, chiefly K-means and BSCAN, are not given any previous information on the structure of the data and its classes. These algorithms are thus trained on the whole dataset to identify clusters of data points with similar characteristics and also to detect any outliers. In the context of our paper, clustering algorithms are useful for both binary and multi-class IDS problems because they can be used to discover hidden patterns for different types of traffic without the need for human intervention.

B.1 | K-means clustering

K-means clustering is a simple unsupervised algorithm that groups data points in a number of clusters specified as input. From a practical standpoint, a datapoint is simply a vector in some high-dimensional space whose cluster is, at least initially, randomly assigned and based on these assignments, the centroids of the clusters are computed. The latter are meant to represent the mean of all instances of vectors within a given cluster. The algorithm continuously assigns each datapoint to the cluster represented by the nearest centroid and then moves the centroid to the position of the new mean of the cluster. Eventually, the centroids converge to or fluctuate around some value (within a given error bound) and thus cluster classifications remain stable.
Because k-means uses the Euclidean distance to the nearest centroid as the proxy for determining what cluster a datapoint belongs to, it assumes that the clusters are convex shaped, such as circles or ellipses. To account for the possibility of an arbitrary number of non-convex clusters, we also tested the DBSCAN algorithm.

B.2 | DBSCAN

Like k-means, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) works by partitioning datapoints to determine cluster boundaries. However, this algorithm requires no input regarding the number of clusters to find. Instead, it first identifies core points as datapoints that are within a certain radius of several other datapoints in multi-dimensional space. Groups of core points are considered areas of high density, and each group forms the center of a separate cluster. At the edges of the clusters, the core points may be within radius of other datapoints that are still part of the cluster, even though they aren't within radius of enough other points to be considered core points themselves. All other datapoints which don't connect to a core point are considered outliers.
A list of the main parameters to consider when using DBSCAN are:

- eps: the radius around a datapoint within which to search for other datapoints. This is a very important parameter for DBSCAN, as it determines the distinction between core and non-core points, significantly influencing the results of clustering.
- min_samples: the number of datapoints that must be within radius of a datapoint for it to be considered a core point - also a significant determinant of clustering results.
- metric: the method used to calculate distances such as radiuses in multi-dimensional space, which is set by deault to the euclidean metric.

C | Evaluation metrics

Accuracy and recall (especially of the attack class, in our case) are often the most useful metrics for comparing the performance of different algorithms. In binary classification, accuracy refers to the ratio of datapoints correctly identified as their true category (either normal or attack, in our case) over the total number of traffic instances in the dataset:

$$\text{Accuracy} = \frac{\text{TrueNegatives} + \text{TruePositives}}{\text{TruePositives} + \text{FalsePositives} + \text{TrueNegatives} + \text{FalseNegatives}}$$

Precision is instead the ratio of correctly predicted traffic under a certain category (normal or some type of attack) over the total number predictions made for that category. Instead, recall is the ratio of correctly predicted traffic under a certain category to the total number of datapoints that actually fit into that category.

$$\text{Precision} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}} \qquad \text{Recall} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives}}$$

Finally, the F1-score refers to the harmonic mean of precision and recall:

$$\text{F1} = \frac{2 \text{ x Precision x Recall}}{\text{Precision} + \text{Recall}}$$