DDoS client and attacker detection, using One Class SVM

Guilherme Pereira
DETI Aveiro
University of Aveiro
Aveiro, Portugal
guilherme.pereira@ua.pt

José Costa
DETI Aveiro
University of Aveiro
Aveiro, Portugal
guilherme.pereira@ua.pt

***Abstract*—In the current landscape of the internet Distributed denial of service (DDoS) attacks are quickly becoming one of the most prevalent types of cyber security threats. In this work we propose a machine learning based approach not to identify that a DDoS attack is occurring, but rather to distinguish clients from attackers with the objective of blocking the attackers and white listing the clients during theses attacks. We also go through how we reproduced, gathered and processed the data used on our work as well as explaining how we used two different approaches for the detection.**

I. Introduction

Denial of service is a type of network attack that seeks to make a network resource unavailable to its users by disrupting the services or hosts connected to a network through various types of attack traffic. A Distributed denial of services has the same objective but the incoming attack traffic comes from many different sources usually the attack sources are hosts affected with malware to form what is known as a bot net.

In the last quarter of 2021 it was reported by Cloudflare, one of the most known companies that offers security services in the internet, that in the last half of 2021 a surge of Distributed Denial of Service attacks was observed over the Cloudflare network, it is reported that in just December 32% of their customers reported being target by a ransom DDoS attack and threats. [1]

Our hypothesis is that the usual client generates traffic that can be distinguished from the traffic that is generated by infected hosts taking into account that the the attack traffic can take on many different forms. With that in mind the objective of our project is to train a an anomaly detection algorithm that will spot out anomalies in client traffic, DNS attack traffic, TCP SYN flood traffic, ICMP flood traffic, UDP flood trafic and HTTP request flood traffic. With the right selection of features and algorithm could achieve a good accuracy in identifying attackers traffic as an anomaly.

The bellow sections cover topics such as some related work, how we generated the data, how we processed the data, how we used One class SVM and Logistic Regression.

II. Related Work

In the paper "A Machine Learning Approach for DDoS Detection on IoT Devices" [2] we took inspiration on which types of attacks should be used in this project and decided to choose a subset of the attacks mentioned by the authors. (https://arxiv.org/pdf/2110.14911.pdf).

For the development of the DDOS attack tool, the paper D-SCAP: DDoS Attack Traffic Generation Using Scapy Framework." [3] was a big starting point, to both select the Scapy framework as the main source of the attacks used and how to perform them in a correct manner. (https://link.springer.com/chapter/10.1007/978-981-13-1882-5_19)).

III. Threat Model

As DDoS attacks have evolved over the years, using new techniques and methodologies, a systems infrastructure can be affected in multiple different ways according to the attackers objective.

For this exact reason a Threat Model was elaborated to specify the ideal setting for which the machine learning model was designed and developed.

Taking into account the general infrastructure of a small company, the threat model presented in figure 1 was designed. It contains the company firewall that monitors the traffic between the outside and the inside company network, the internal routing for the devices communication inside the company represented by a router in this instance and a small data center containing a common Apache HTTP server and a BIND9 server.

This model also illustrates the considered attacks that are the TCP SYN, ICMP, UDP, HTTP GET and DNS request floods. These are furthermore explained in the Data Generation subsection.

The machine learning model presented in this project can be used to detect one, or multiple attacks that are being conducted on a server, by constantly monitoring the running services and blocking the source ips that may be producing an attack at a firewall level, so as to not affect the running service.

IV. Dataset

A huge obstacle in this project is the creation of a dataset suitable for the problem, as there are no available data sets online that contained both client data and the various different DDoS attack data that we needed.

In this section we will demonstrate and explain how we set up a test environment that allowed us to perform different

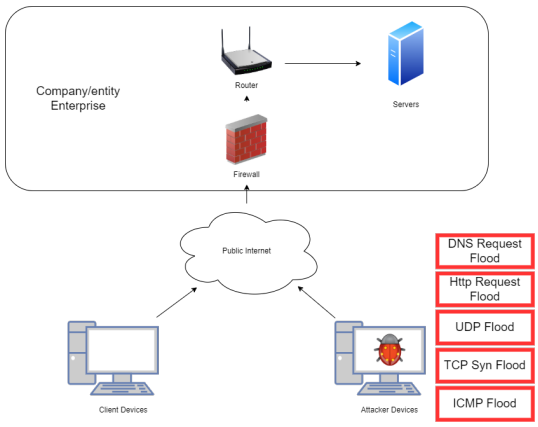

Fig. 1. Threat Model Example

types of DDoS attacks and capture that data, as well as how we simulated and captured the needed network information of a regular client on the internet.

A. Data Sources

1) Client Data: As the targets of DDoS include multiple services and modules, we need to find a way to capture information regarding of the network traffic but also information on the DNS servers and Apache servers that regular company would have and could be targets of DDoS attacks.

To capture regular network traffic and DNS specific traffic wireshark, which is a network sniffing program, is used. To get information that is stored on the Apache service logs, since we simulated the client using a regular browser and didn't have access direct access to the actual DNS servers and Apache servers, as explained in the Data Generation subsection.

2) Attack Data: Regarding the attack data, since in this part we developed a test environment we controlled both the Apache server on the "/var/log/apache2" path and DNS server on the "/var/log/bind" path, which means that the data regarding those services could be directly obtained from their respective logs. Wireshark was also used to capture the overall network traffic that the attackers generate.

B. Data Generation

1) Client Data: To acquire regular users data to train the machine learning model, information from both the web server, dns server and wireshark has to be aquired. This requires considerable usage from regular users on both of the services, so that information can be logged and used.

As for the purposes of this project, this infrastructure doesn't exist and the services are not available, so a simulation of both the Apache and BIND9 logs was required.

To accomplish this goal we used a Ubuntu 18.04 virtual machine for a controlled simulation environment, but data captured from a regular device should also be valid.

To simulate the Apache logs from a web server, the browser network tools were used to record all logs from visited websites.

Wireshark was used to record all network information and furthermore extract the DNS resolutions, simulating the DNS logs.

The choice of the browser was a tricky decision. The browsers tested were Chrome, Firefox and Edge. These were tested in different operating systems, for registering the logs using the previously mentioned network tools, but, for both Chrome and Firefox browsers, this logging functionality was malfunctioning and in most instances all logs were lost or saved with 0 bytes.

For this reason the Edge browser was chosen, most problems with logging did not occur and the format generated contained all relevant information used for this project.

With all of the setup running and monitored as shown in figure 2, the next steps were to simply browse over the internet as a regular user would, this includes, perform searches, watching videos, reading online journals etc. As this process requires a real user to browse the web, the data is harder to acquired compared to the attack data.

When this acquisition process is over, 2 files will be generated:

- A .har file with the simulated Apache logs from the browser
- A .pcap file containing all the information from the packet capture (including the dns information)

Both these files are eventually used to extract metrics, as explained in the Data Processing section.

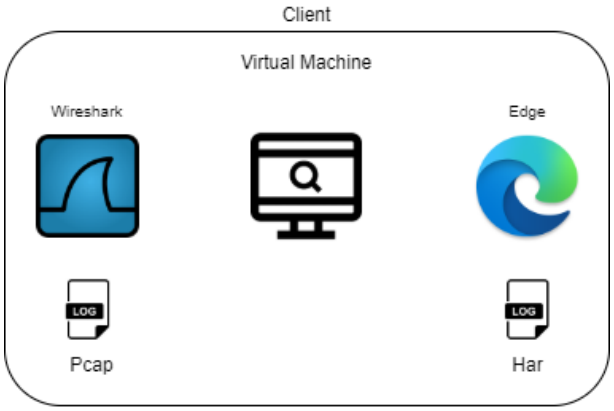

Fig. 2. Client Simulation Setup

2) Attack Data: To acquire attack data to train the machine learning model, this process requires an elaborated simulation environment when compared to the client data generation, as a web server can be simulated, and the attacks can be performed on this server. For this a Apache web server and the bind9 dns services will be deployed. This allows for a more realistic scenario when retrieving the attack information.

To accomplish this goal we used a VirtualBox to create 2 virtual machines and a virtual network between them, as shown in picture 3. The first virtual machine is running Ubuntu 18.04 used to simulate the data center, with the Apache [4] and DNS (Bind9) [5] service installed.

The second virtual machine is running Kali Linux 2021.4a and is used to attack the server with a custom ddos python script.

This python script is responsible to perform the 5 types of attacks on a server, using mainly the scapy library [6]. All of these attacks can be performed in a constant request loop with a custom time range, that for this particular data set was between 10ms and 1s. In more detail, each attack is explained as follows:

- **TCP SYN:** This attack is performed by sending multiple custom TCP packet with the SYN flag.
- **UDP:** This attack is performed by sending multiple custom empty UDP packets.
- **ICMP:** This attack is performed by sending multiple ICMP packets.
- **GET:** This attack targets specifically the Apache server by sending multiple GET requests to multiple pages of the server with a custom fixed header, but requesting different pages that are randomly chosen from a list of 4 possibilities.
- **DNS:** This attack targets specifically the BIND9 DNS server by sending multiple DNS query to the dns server, requesting different name resolutions chosen randomly from a list of 4 possibilities.

This application is also capable of forging the source ip's of the packets for every attack, allows for multiple parameterization and threading to perform a DDOS.

With all of this setup, the first step to collect data is to clear both the bind9 and Apache logs, capture network traffic on the virtual network between the VM's with wireshark (on the VM with the services) and lastly execute the desired attack with the virtual machine responsible for the attack with the custom python DDOS tool.

When this acquisition process is over, 3 files will be generated:

- A .pcap file containing all the information from the packet capture
- A bind.log file containing the Bind9 DNS log.
- A apache.log file containing the Apache log.

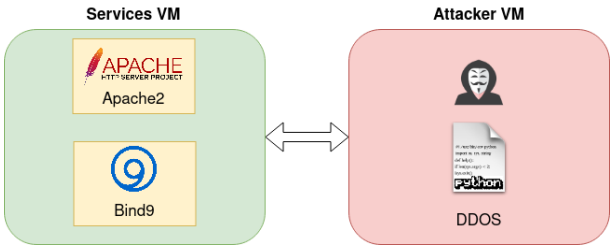

Fig. 3. Client Simulation Setup

V. Data Processing

1) Metrics Extraction: As the multiple data logs contain a lot of information, certain metrics were chosen that will derive the features used to train the machine learning model.

To accomplish this goal 2 python scripts were developed, that take as input the logs from either the client data or the attack data, every 1 second.

The client data script takes as input the .har file with the simulated Apache logs from the browser and the .pcap file containing all the information from the packet capture. The attack data script take as input the .pcap file containing all the information from the packet capture, the bind.log file containing the Bind9 DNS log and the apache.log file containing the Apache log.

Both of these generate a csv file with the metrics in the exact same format, so that features can be extracted in the same manner, independently of the data sources used.

All of the metrics are presented with the correspondent data source in the following table:

Metric	Client Data	Attack Data
No. Packets	.pcap	.pcap
Packet Size Sum	.pcap	.pcap
No. TCP Packets	.pcap	.pcap
No. UDP Packets	.pcap	.pcap
No. ICMP Packets	.pcap	.pcap
No. Apache Req.	.har	apache.log
No. Apache Errors	.har	apache.log
Apache Logged in	.har	apache.log
No. Different Pages Accessed	.har	apache.log
No. DNS Queries	.pcap	bind.log
No. DNS Errors	.pcap	bind.log

2) Features Extraction: After processing all raw data from the log files to the selected metrics, as a csv file, feature extraction is used to calculate statistics over a period of 10 metrics measurements (10 seconds).

A python script was developed that generates a final csv file containing the mean, max and min for the number of packets, packet size, number of TCP, UDP and ICMP packets, number of Apache requests and errors, number of DNS requests and errors. Adding to these features, the Apache logged in state, the time variance between requests and the relative distance between the source ip and the server location were aso used.

All of these features are used to train the machine learning model for both the anomaly cases and the regular use case.

VI. Methodology

A. First approach using One class SVM

When we first started working on our project, even before gathering the data, we had in mind that the number of anomalies that we would have would be relatively small compared to the normal data, so, we started looking up at some anomaly detection algorithms. Although as explained above this ended up not being true, as it is more easy for us to generate attack traffic than normal traffic. After having the data we out of curiosity still wanted to apply a anomaly detection algorithm to our project since the type of anomalies that we generated do not represent all of the existing DDoS type attacks.

The selected algorithm for the task was One class SVM which is an unsupervised machine learning model used in anomaly. One class SVM as SVM is used to separate two classes however instead of using a hyper plane to separate them, One class SVM uses a hypersphere instead.

The first step that had to be done was to split all of our data into a train, validation and test sets. The training set containing 80% of all the non anomalous data and the validation and test sets containing 10% of the non anomalous data and an equal amount of anomalies. The number of non anomalous data used was 829.

To train and validate the model we only used one type of anomaly in this case DNS request flood.

Taking into account the origin of the data our hypothesis was that our 31 features were highly correlated, so we could reduce the number of features without loosing to much variance.

To reduce the number of features we used Principal component analysis on all of the sets. Before applying principal component analysis it is important to first normalize the data, by scaling each feature to a given range, in our case by using MinMaxScaler from the sklearn machine learning library, and to standardize the features by removing the mean and scaling to unit variance, which can be achieved by using sklearn StandardScaler. [8] [9]

With principal component analysis we were able to reduce the number of features (31) to the smallest number that kept 99% of the variance (16).

```
[0.35913102 0.48000944 0.58008771 0.65385948 0.7263564  0.78002419
 0.8264617  0.86578147 0.90112002 0.93114961 0.95423544 0.96691846
 0.97758213 0.98342598 0.98800228 0.99111908 0.99322836 0.99480583
 0.9962215  0.99762226 0.99885964 0.99963848 0.99993322 0.99998994
 0.99999972 0.99999995 1.         1.         1.         1.
 1.        ]
```

Fig. 4. Variance kept by each one of the 31 features

With all of this we used the validation set to obtain the best hyper parameters for One Class SVM. The hyper-parameters of the One class SVM include:

- **Nu:** which is an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. the nu values are in range (0, 1].
- **Kernel:** Specifies the kernel type to be used in the algorithm. Takes on the following values [‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’]
- **Gamma:** Kernel coeficient. Which can be ’scale’ or ’auto’.

To get the best hyper parameter we used F1 score as an indicator as we go through every combination of the above hyper-parameter values. After running the hyper parameter optimization the hyper-parameters that gave the best F1 score were the ones in figure 5.

After obtaining the best hyper parameters we trained One class SVM with those parameters to get the ”final model” and tested it with the test set were a F1 score of 1 was obtained.

```
best nu:  0.03
best kernel:  sigmoid
best gamma:  scale
best F1:  0.9879518072289156
```

Fig. 5. Hyper parameters that give the best F1 score in validation

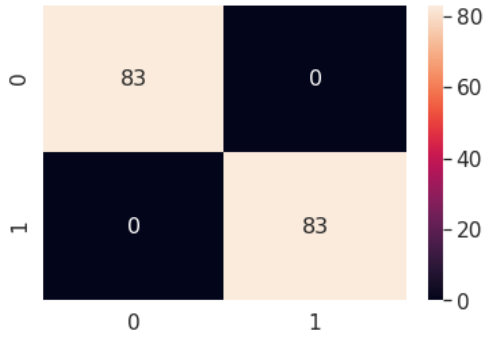

Fig. 6. Confusion matrix of predict using the test set

Then after having tested using the test set we wanted to verify how the model would react to the other four types of attacks that were not used to validate and test. So to do just that we gathered some extra non anomalous data that the model also did not use to train and validate and attached it to the different anomalies. The values obtained for the other types obtain go as follow:

- **ICMP attacks** A F1 score of 1 was obtained testing for a dataset containing 96 anomalies and 96 non anomalies.
- **GET attacks** A F1 score of 0.89 was obtained testing for a dataset containing 96 anomalies and 96 non anomalies.
- **UDP attacks** A F1 score of 1 was obtained testing for a dataset containing 96 anomalies and 96 non anomalies.
- **TCP attacks** A F1 score of 0.93 was obtained testing for a dataset containing 96 anomalies and 96 non anomalies.

B. First approach with k-fold

The first implementation had a problem, the problem being that the values mentioned previously were not obtained every time that we trained, validated and tested the model. After spending so time to try and figure out why that was happening noticed that the problem came from the fact that the data split into train set, test set and validation set was not always the same. To obtain the previously mentioned values we had to search for the seed of the splits that gave us the best results. So now that we knew that the split was a factor that directly impacted the best hyper parameters we decided to implement k-fold cross validation which is used to evaluate a machine learning model on limited data samples. [7]

To perform the k-fold validation we used sklearn k-fold with shuffle and n_splits equal to 5. for each split we now had 80% of the non anomalous data in the training set and 20 % for the test set. To obtain the validation set we sliced the test set in half. So at the beginning of each k-fold split we had 80% of non anomalous data in the training set, 10% on the test set and 10% in the validation set. To the validation and test set we also appended an equal length number of anomalies. Figure 7 can be used to better understand how we split the data in each k-fold iteration.

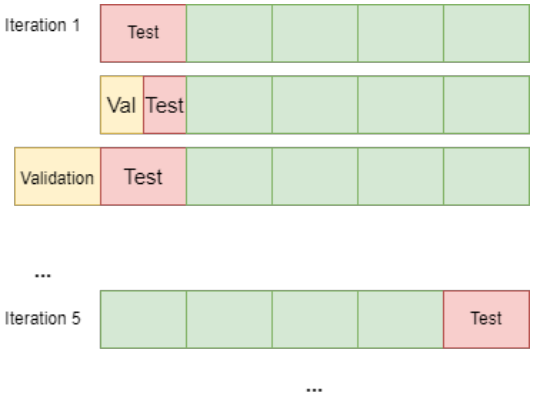

Fig. 7. k-fold splits

With our data-set this translated to having for each split a training set of 663 non anomalies, a test set with 83 anomalies and 83 non anomalies and a validation set with 83 anomalies and 83 non anomalies. So now for each split of k-fold we perform all of the previously mentioned steps and save the best configurations and score. At the end of the 5 splits we have for each split the F1 score using the test set of the split, which we then use to obtain the average F1 score of the 5 iterations.

To test the algorithm to different types of attacks as we previously did we saved the model that had the best F1 score out of all of the splits.

Bellow we demonstrate for each k fold iteration the F1 score obtained, the mean F1 score of all the iterations as well as the best configuration obtained.

```
Best F1 score:  1.0
Best Nu:  0.01
Best_kernel:  sigmoid
Best_gamma:  scale
scores:  [1.0, 1.0, 0.976470588235294, 1.0, 0.993939393939394]
Mean F1 score:  0.9940819964349377
```

Fig. 8. k-fold splits

At the end we ended up having the best F1 score being 1 and the mean being 0.99.

We then tested the best model out of the five iterations on the other four types of attacks that the model as never seen.

- **ICMP attacks** A F1 score of 1 was obtained testing for a dataset containing 96 anomalies and 96 non anomalies .
- **UDP attacks** A F1 score of 0.99 was obtained testing for a dataset containing 96 anomalies and 96 non anomalies .
- **HTTP attacks** A F1 score of 0.69 was obtained testing for a dataset containing 96 anomalies and 96 non anomalies .
- **TCP attacks** A F1 score of 0.74 was obtained testing for a dataset containing 65 anomalies and 96 non anomalies.

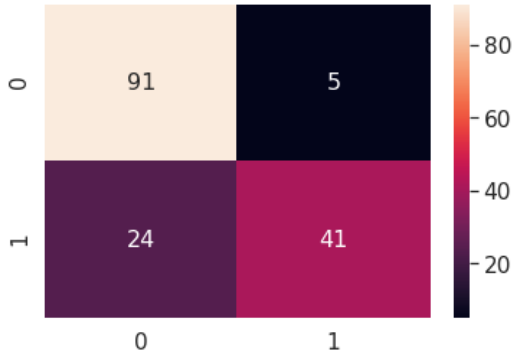

Fig. 9. Confussion matrix of the TCP attacks

The values in general in k-fold go accord to the values found in our first approach.

C. Second approach using Logistic Regression

Since it was more easy for us to generate anomalous data, we decided that it would be interesting to see how a supervised algorithm, Logistic Regression, would perform. So to do just that we appended all different types of anomalies and non anomalies into a single labeled array (size of 1658) where for anomalies the label was equal to one, for non anomalous data the label was equal to 0. Then by using the same approach with k-fold for each split we had 80% of all data in the training set, 10% in the test set and 10% in the validation set. Note that this time did not apply principle component analysis on the data, only data normalization with MinMaxScaler and StandardScaler. For the hyper parameters we took into account:

- **Solver:** Algorithm to use in the optimization problem, the values it can have include ['newton-cg','lbfgs','sag','saga'].
- **penalty:** Specifies the norm of the penalty. the values it can have include ['l2','none'].
- **multi_class:** The values we took into account include ['ovr','auto'].

After performing the k-fold cross validation with Logistic regression the values obtained were the following:

As we can see it the scores did not vary much from the One class SVM. However let us now in the next section compare One class SVM with Logistic Regression.

D. One class SVM vs Logistic Regression

Overall one class svm and logistic regression did not vary that much in terms of performance. However after finishing all

```
Best F1 score:  0.9940119760479043
Best penalty:  l2
Best solver:  newton-cg
Best multi_class:  auto
scores:  0.9793103448275862 0.9826589595375723
scores:  0.9864864864864865 0.9940119760479043 0.9655172413793104
Mean F1 score:  0.9815970016557719
```

Fig. 10. Results after performing k-fold with logistic regression

of the above, we had some time to perform one last additional task. We were able to generate more complex DDoS attacks, what we mean by this is that we were able generate attacks that were more difficult to distinguish from a normal client, by adding mor threads to the attacker's nodes and vary the send ratio of the attack traffic. Using this new complex data to test both the one class svm model and the logistic regression one we noticed that the one class svm performed better than the logistic regression one, which makes sense as it is an anomaly detection algorithm and logistic regression did not use such complex attacks to train.

Bellow the F1 score and the confusion matrix are showcased for the results obtained by using a set with 96 non anomalies and 96 complex HTTP request flood attack traffic for One class SVM and Logistic Regression.

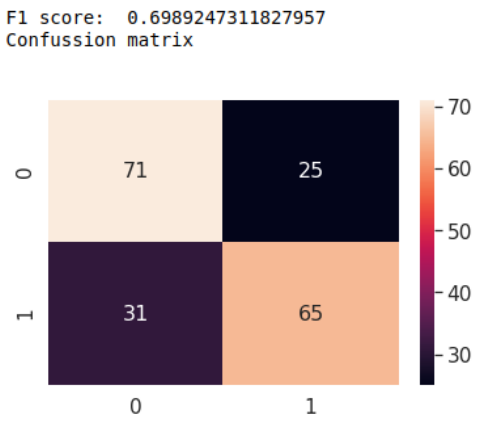

Fig. 11. Results after performing k-fold with logistic regression

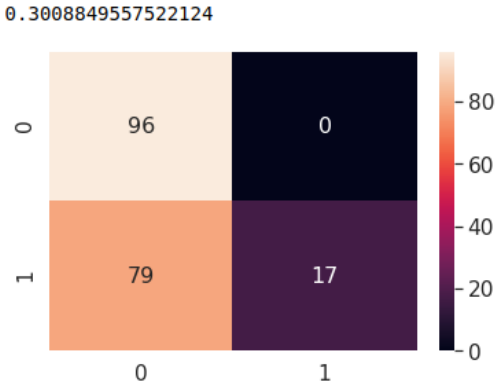

Fig. 12. Results after performing k-fold with logistic regression

VII. Using the models to distinguish clients from attacks

After selecting and training one of this models we now can have them look at data generated from different IP addresses, and for instance by defining a threshold for the number of attack traffic that the models identify we can then either black list the client or white list it.

VIII. Future improvements

To finish of this report we would like to point out some things that need to be improved and could be done in the future with this work. Regarding the data, the data set was relatively small so having more data would prove to be beneficial to our work, furthermore by using real information from a real web server to gather regular use and attack information would improve the data quality. The data extraction and feature selection process could perhaps be different and improve the results. To make our first approach more viable we would also have to search for the best seed that could give a better over all performance on our data.

References

[1] Cloud flare DDoS trends". [Online], Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/. [Accessed Jan. 24, 2022].

[2] A Machine Learning Approach for DDoS Detection on IoT Devices [Online], Available: https://arxiv.org/pdf/2110.14911.pdf [Accessed Dez. 20, 2021].

[3] D-SCAP: DDoS Attack Traffic Generation Using Scapy Framework [Online], Available: https://link.springer.com/chapter/10.1007/978-981-13-1882-5_19 [Accessed Dez. 20, 2021].

[4] Apache documentation [Online], Available: https://httpd.apache.org/docs/2.4/ [Accessed Jan. 7, 2022].

[5] Bind9 documentation [Online], Available: https://bind9.readthedocs.io/en/latest/ [Accessed Jan. 7, 2022].

[6] Scapy documentation [Online], Available: https://scapy.readthedocs.io/en/latest/ [Accessed Dec. 27, 2021].

[7] Towards data science k-fold-cross-validation". [Online], Available: https://towardsdatascience.com/k-fold-cross-validation-explained-in-plain-english-659e33c0bc0. [Accessed Jan. 24, 2022].

[8] Sklearn Standard Scaler". [Online], Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html . [Accessed Jan. 25, 2022].

[9] Sklearn MinMaxScaler ". [Online], Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html . [Accessed Jan. 25, 2022].