

ML FOR DDOS DETECTION

Nick Sona

+

•

Problem Definition

- DDoS attacks pose a large threat against an organizations network security & integrity
 - DDoS attacks work by flooding a targeted server, service, or network by flooding it with a massive volume of internet traffic. This flood of traffic consumes the target's resources and makes it unable to respond to legitimate requests. DDoS attacks pose a significant threat to network security due to their potential to cause financial loos, disrupt business operations, and create opportunities for further attacks.
- Network security administrators need a way to automate the detection of DDoS attacks
 - Impractical to sift through thousands of packets and analyze them manually
 - Not enough time
 - Impractical to hire someone dedicated to analyzing network traffic when it can be automated
 - Human error
- This project addresses the need for automated DDoS detection by implementing a machine learning model capable of detecting multiple different types of DDoS attacks by analyzing network traffic.



Solution Overview

In this project I used the Python Scikit-learn and Scappy libraries to train a machine learning model on various Wireshark .pcap files that represent DDoS attack traffic for the following network communication protocols: DNS, LDAP, NTP, SNMP, SSDP, and UDP. Upon completion of training & testing, the model was proven to be effective in detecting DDoS traffic for the mentioned protocols.

Methodology Workflow:

1. Data Preprocessing
2. Feature extraction
3. Model Selection
4. Model Training
5. Model Testing
6. Evaluation



Data Preprocessing

1. Initial datasets included 8 pcap files: benign.pcap, DNS.pcap, LDAP.pcap, NTP.pcap, SNMP.pcap, SSDP.pcap, UDP.pcap, UDP-lag.pcap
2. Initial datasets were transformed to be only 500 packets long each using editcap command line tool

```
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\benign.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\benign_500.pcap" 1-500
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\DNS.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\DNS_500.pcap" 1-500
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\LDAP.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\LDAP_500.pcap" 1-500
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\NTP.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\NTP_500.pcap" 1-500
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\SNMP.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\SNMP_500.pcap" 1-500
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\SSDP.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\SSDP_500.pcap" 1-500
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\UDP.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\UDP_500.pcap" 1-500
PS C:\Program Files\Wireshark> ./editcap -r "C:\Users\NickSona\Desktop\CSC5290\dataset\UDP-lag.pcap" "C:\Users\NickSona\Desktop\CSC5290\dataset\UDP-lag_500.pcap" 1-500
```

3. Next, each file was ingested by the DataPrep&Extract.py script and split into two datasets each 250 packets in length by the read_and_split_pcap() function. One dataset reserved for training the model and another for testing.

Feature Extraction

In this stage, features were extracted from each newly created 16 dataset's packets. This was accomplished with the following functions in DataPrep&Extract.py:

- `extract_features()`
 - This function works by iterating through each packet in the dataset and extracting the following features:
 - Packet size
 - Protocol
 - Source IP
 - Destination IP
 - Flags
 - Timestamp
 - After extraction, the features for each packet are added to a dataframe in python that represents the one of the 16 datasets.
- `calculate_frequency()`
 - This function works by grouping packets with the same features and calculating their frequency by dividing the count of the similar packets by the time duration of the entire packet sequence for the 250 packets. The frequency for each packet group is appended to the dataframe from `extract_features()`

Feature Extraction Cont.

- `main()`
 - This function works by iterating through each 500 length pcap file and:
 - Calling `read_and_split_pcap()` to split the file into two datasets
 - Calling the `extract_features()` function for the two datasets and writing the results to two dataframes (`train_df` and `test_df`)
 - Calling the `calculate_frequency()` function for the two dataframes and writing the results to each dataframe
 - Exporting the two dataframes to csv for visibility
- Upon completion of the `DataPrep&Extract.py` program, a total of 16 .csvs are generated that include packet features for each packet for each of the 16 datasets created in the Data Preparation stage.



















Feature Extraction

.csv Example:

	A	B	C	D	E	F	G	H	I	J
1	Packet Size	Protocol	Source IP	Destination IP	Flags	Timestamp	Frequency			
2	60	17	174.16.0.5	192.168.50.1		1543677760.004519	21.66203431283564780616881015			
3	60	17	174.16.0.5	192.168.50.1		1543677760.004521	21.66203431283564780616881015			
4	60	17	174.16.0.5	192.168.50.1		1543677760	21.66203431283564780616881015			
5	60	17	174.16.0.5	192.168.50.1		1543677760.005812	21.66203431283564780616881015			
6	60	17	174.16.0.5	192.168.50.1		1543677760	21.66203431283564780616881015			
7	60	17	174.16.0.5	192.168.50.1		1543677760.031652	21.66203431283564780616881015			

Total .csv list:

 benign_testing	11/9/2024 6:30 PM	Microsoft Excel Com...	21 KB
 benign_training	11/9/2024 6:30 PM	Microsoft Excel Com...	21 KB
 DNS_testing	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 DNS_training	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 LDAP_testing	11/9/2024 6:31 PM	Microsoft Excel Com...	20 KB
 LDAP_training	11/9/2024 6:31 PM	Microsoft Excel Com...	20 KB
 NTP_testing	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 NTP_training	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 SNMP_testing	11/9/2024 6:31 PM	Microsoft Excel Com...	20 KB
 SNMP_training	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 SSDP_testing	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 SSDP_training	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 UDP_testing	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 UDP_training	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 UDP-lag_testing	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB
 UDP-lag_training	11/9/2024 6:31 PM	Microsoft Excel Com...	21 KB

Model Selection – Random Forest

- What is Random Forest?
 - A machine learning algorithm that builds multiple decision trees during training and combines the outputs to improve predictive performance
 - Each tree is trained on a random subset of data
 - A random subset of features is considered at each split in the tree
 - The final prediction is made by the most common classification among the trees
- Why Random Forest?
 - High Accuracy for high dimensional data, making it suitable for detecting complex patterns in DDoS attack traffic
 - Robustness – reduces risk of overfitting by averaging predictions across multiple decision trees
 - Handles both numerical and categorical data, which makes it suitable for packet features like packet size, IP, and protocol type



Training & Testing

1. Feature Preparation

- `Ip_to_int()` Function converts IPs to integer values to make them machine readable
- The `Flags` column is encoded numerically using the `LabelEncoder` class

2. Data Preprocessing

- Each of the sixteen datasets are loaded and labeled 0-7 (`benign_train = 0` & `benign_test = 0`)
- Each dataset is then concatenated into a single training / testing dataset

3. Defining Features and Labels

- Features are then assigned to variables `X_Train` / `X_Test` to hold and include numerical representations of all features extracted in the feature extraction stage
- Labels (0-7) are then assigned to `Y_Train` / `Y_Test` hold the traffic classifications (benign, DNS, etc)

4. Random Forest Training

- The random forest classifier is set to 100 trees, max depth of 10, and Gini impurity (for splitting criterion)
- Each tree is trained on a random subset of the training data with a random subset of features at each split



Testing Cont.

- Testing Process
 - After training, the model was evaluated using unseen data (X_{test}), and predictions were generated using the `predict()` method of the Random Forest Classifier.
 - The actual labels (y_{test}) were compared to the predicted labels (y_{pred}) to assess performance.
- Evaluation Metrics:
 - Accuracy Score: Measures the percentage of correctly classified traffic. Calculated using `accuracy_score(y_test, y_pred)`.
 - Classification Report: Provides precision, recall, F1-score, and support for each class, offering a detailed breakdown of model performance.

```
# Make predictions and evaluate the model
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print(classification_report(y_test, y_pred))
```

Results

```
benign_test['Label'] = 0
dns_test['Label'] = 1
ldap_test['Label'] = 2
ntp_test['Label'] = 3
snmp_test['Label'] = 4
ssdp_test['Label'] = 5
udp_test['Label'] = 6
udp_lag_test['Label'] = 7
```

Model Accuracy: 93.70%

	precision	recall	f1-score	support
0	1.00	1.00	1.00	250
1	0.66	1.00	0.80	250
2	1.00	1.00	1.00	250
3	1.00	1.00	1.00	250
4	1.00	0.50	0.66	250
5	1.00	1.00	1.00	250
6	1.00	1.00	1.00	250
7	1.00	1.00	1.00	250
accuracy			0.94	2000
macro avg	0.96	0.94	0.93	2000
weighted avg	0.96	0.94	0.93	2000

Evaluation

- Accuracy: The model achieved an overall accuracy of 93.70%, showing that it correctly classified most traffic types.
- Precision, Recall, F1-Score:
 - Benign (0), DNS (2), LDAP (3), and others achieved perfect precision, recall, and F1-scores of 1.00.
 - Class 4 (SNMP) showed lower recall (0.50) and F1-score (0.66), suggesting difficulty identifying patterns in this traffic type.
 - Macro Average: Evaluates performance across all classes equally, yielding Precision: 0.96, Recall: 0.94, F1-Score: 0.93.
 - Weighted Average: Adjusts metrics based on number of instances, which gives the same values since the datasets were balanced (250 each)
- Overall, these results show that the model is considerably effective in identifying whether network traffic is benign or malicious across various networking protocols