METHODS OF DETECTIONS IN A DDOS ATTACK

IMPLEMENTATION REPORT

CYBERSECURITY

BCSE410L

E1+TE1

Under

SAIRABANU J

submitted by

PRAMITI SHEKHAR SINGH 21BCE3479

ROHAN BANSAL 21BCE2443

SHIVAM DEY 21BCE2720

in partial fulfilment for the award of the degree of

BACHELOR'S IN TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

MAY 2024

INTRODUCTION

Distributed Denial of Service (DDoS) attacks represent a pervasive and significant threat to cybersecurity, primarily targeting the availability of online services, websites, and applications. These attacks aim to overwhelm a target system by generating a flood of malicious traffic, often originating from a large network of compromised devices (known as a botnet). By inundating the target with this excessive traffic, DDoS attacks exhaust server resources, making services inaccessible to legitimate users and causing substantial disruptions.

As DDoS attacks have evolved, attackers have employed increasingly sophisticated techniques that can evade traditional detection methods, such as signature-based detection. Signature-based approaches, which rely on identifying known attack patterns, are often unable to keep pace with the rapid development and variation in DDoS strategies. This inadequacy has led to a heightened need for advanced, proactive detection techniques capable of recognizing and mitigating diverse DDoS attack patterns.

Advanced DDoS detection techniques such as **anomaly-based**, **behavioral-based**, and **signature-based detection** offer a more robust defense by identifying deviations from normal traffic patterns, analyzing user behavior, and leveraging real-time signal processing techniques, respectively. Each method provides unique capabilities to detect, analyze, and mitigate DDoS attacks effectively. These techniques not only enhance detection accuracy but also facilitate faster identification of unusual traffic, thereby minimizing the time and impact of DDoS incidents.

We explore each of these detection techniques in detail, examining their underlying methodologies, advantages, and limitations in the context of detecting and mitigating DDoS attacks. Through a comprehensive understanding of these advanced detection mechanisms, organizations can better protect their systems and ensure continuous service availability, even in the face of evolving cyber threats.

DESCRIPTION

1. Anomaly-Based Detection

Anomaly-based detection is centered on identifying deviations from typical network behavior. The foundation of this method lies in its ability to define a baseline for what constitutes "normal" traffic patterns and flag deviations as potential threats. It combines statistical methods, machine learning, and dynamic threshold setting to accomplish this.

- **Detailed Baseline Creation:**
  - **Data Collection:** The system collects traffic data over an extended period, capturing attributes such as packet size, rate of requests, user session durations, and connection types. Baseline creation typically requires weeks or months of observation to account for daily, weekly, and seasonal variations.
  - **Data Analysis and Filtering:** Raw traffic data is analyzed and filtered to remove noise (e.g., legitimate spikes during high-traffic events). Filtering ensures that only normal behavior is reflected in the baseline.
  - **Baselining Algorithms:**
    - *Statistical Models:* Simple statistical methods, such as calculating mean, standard deviation, and using moving averages, provide an initial baseline. For instance, a statistical model might calculate the average number of requests per minute and mark deviations beyond three standard deviations as anomalies.
    - *Machine Learning Models:* More advanced anomaly-based detection uses machine learning algorithms like *k-means clustering* to group traffic data into clusters representing "normal" patterns. Clusters far from this norm could indicate potential DDoS traffic.
    - *Time-Series Analysis:* Seasonal and trend analysis, like using Seasonal Autoregressive Integrated Moving Average (SARIMA) models, accounts for expected variations in traffic and helps reduce false positives.
- **Threshold Setting and Dynamic Adjustments:**
  - Static thresholds (e.g., fixed number of requests per second) can be ineffective for fluctuating network traffic. Dynamic thresholds, adjusted based on real-time data, make anomaly detection more resilient to changes.
  - *Example:* During a flash sale, legitimate traffic spikes might otherwise trigger false positives. Dynamic thresholds can increase acceptable traffic limits during these times based on historical data patterns, reducing false alarms.
- **Machine Learning Integration:**
  - **Supervised Learning Models:** Supervised models, trained on labeled data (e.g., known attack vs. legitimate traffic), improve detection by learning specific features associated with attacks.
  - **Unsupervised Learning Models:** Unsupervised learning techniques, such as autoencoders, can identify anomalies without labeled data by compressing data into a lower-dimensional space. Deviations in the reconstruction error of autoencoders indicate potential anomalies.

- **Deep Learning Models:** Complex networks like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) models can analyze traffic flows for abnormalities at a granular level, capturing both spatial and temporal anomalies.

- **Advantages and Limitations (Expanded):**
  - **Advantages:**
    - Highly adaptable, making it useful for new and sophisticated attacks.
    - Able to detect "low and slow" DDoS attacks that gradually increase traffic to avoid detection.
  - **Limitations:**
    - Requires significant computational resources to maintain real-time thresholding and analyze deviations.
    - Sensitive to network changes (e.g., new users or services), leading to recalibration needs and potential temporary false positives.

2. Behavioral-Based Detection

Behavioral-based detection is especially useful for identifying anomalies at the user or application level, as it focuses on typical interactions and usage patterns. This method relies on observing entities within the network (like users or devices) over time to build behavioral profiles.

- **Behavioral Profiling Process:**
  - **User Profiles:** Profiles for individual users are built by tracking login times, session durations, frequency of page visits, and other usage patterns. For example, a user may normally access specific files between 9 a.m. and 5 p.m. If they suddenly start downloading large amounts of data at odd hours, it could indicate an attack.
  - **Device Profiles:** Devices (e.g., IoT devices) are profiled based on their usual behavior. An IoT thermostat, for instance, should not be transmitting large amounts of data, so abnormal communication from this device might signify a DDoS attempt.
  - **Application Profiles:** Profiles are created for applications based on their typical requests, responses, and session patterns. Web applications, for instance, can have a model based on typical user request rates and navigation patterns.
- **Behavioral Analysis and Deviation Detection:**
  - Behavioral-based detection compares real-time interactions with stored behavioral profiles. Deviations from these profiles are analyzed for potential attacks.
  - **Sequence Analysis:** This approach considers the order of requests. Attackers may mimic legitimate users but fail to follow the exact sequences. For instance, a legitimate user might always visit a "Home" page before a "Checkout" page, while a bot might skip this sequence.
  - **Markov Chains and Probabilistic Models:** Markov models can analyze likely transition states between requests. A sudden increase in the likelihood of unusual transitions can flag a DDoS attack.
  - **Feature Engineering:** Traffic features (like response times and request types) are extracted, and feature selection methods (such as Principal Component Analysis) reduce data complexity, enabling accurate anomaly detection.
- **Advanced Machine Learning Techniques:**

- **Recurrent Neural Networks (RNNs):** RNNs are particularly useful as they can capture sequence data, enabling the detection of patterns over time.
- **Clustering and Outlier Detection:** Clustering algorithms, like *DBSCAN* (Density-Based Spatial Clustering of Applications with Noise), can identify abnormal user or device behaviors as outliers.

- **Advantages and Limitations (Expanded):**
  - **Advantages:**
    - Can detect complex, low-and-slow DDoS attacks that mimic normal users' behaviors.
    - Effective for preventing account abuse, credential stuffing, or bots that slowly flood systems.
  - **Limitations:**
    - Behavioral detection is vulnerable to shifts in normal traffic behavior and requires retraining as behaviors evolve.
    - Complex and computationally intensive, especially when tracking numerous unique user behaviors in real time

3. Signature-Based Detection

Signature-based detection is a widely used technique that leverages predefined attack patterns to detect and mitigate known DDoS attacks. It is highly accurate for identifying previously cataloged attacks.

- **Signature Database and Update Mechanism:**
  - **Comprehensive Signature Libraries:** The signature database includes characteristics of various known DDoS attacks, such as TCP SYN floods, UDP floods, and HTTP GET floods. Each signature is crafted based on specific packet structures, flags, sequence numbers, and payload content.
  - **Signature Sources:** Signatures are often obtained from threat intelligence feeds, past incident analyses, and industry-wide databases.
  - **Real-Time Signature Updates:** Security vendors frequently release updates to ensure the signature database reflects the latest attack vectors. Some systems even leverage cloud-based updates that automatically push new signatures across multiple endpoints.
- **Pattern Matching and Analysis:**
  - **Exact Pattern Matching:** Signature-based systems scan network traffic for exact matches to known patterns. For instance, they can detect a SYN flood by looking for a high volume of SYN packets without corresponding ACK responses, which signifies an incomplete TCP handshake.
  - **Partial and Heuristic Matching:** For attacks that don't fully match an existing signature, heuristic methods allow approximate matching by identifying common characteristics of DDoS traffic, such as repetitive payload content or identical packet sizes across requests.
  - **Regular Expression Matching:** For HTTP-based attacks, regular expressions are often used to identify repeated patterns in URLs, parameters, and headers associated with DDoS requests.
- **Advanced Techniques for Enhanced Detection:**

- **Protocol and Packet Analysis:** Some signature-based systems go beyond simple pattern matching by analyzing the structure of network protocols, identifying malformed packets or unusual protocol usage as possible attack signatures.
- **Behavioral Signatures:** Combining signature detection with basic behavioral analysis can help in recognizing attacks that exhibit signature-like behaviors but vary slightly from past patterns. For example, if a new SYN flood variant emerges with minor changes, the system might still catch it by detecting similarities to past SYN floods.
- **Signature Creation for Custom Applications:** Enterprises with custom applications can create tailored signatures to detect DDoS attempts that may not match general attack patterns.

- **Advantages and Limitations (Expanded):**
  - **Advantages:**
    - Quick to detect and respond to familiar threats, making it highly efficient.
    - Well-suited for high-speed, real-time network environments due to low computational overhead.
  - **Limitations:**
    - Cannot detect unknown or variant attacks, as it relies entirely on pre-existing signatures.
    - Requires frequent updates, which may not always be feasible for networks with limited internet connectivity.

When combined, these techniques provide a robust, multi-layered defense:

1. **Initial Screening with Signature-Based Detection:** This layer handles well-known threats with high accuracy and minimal resources.
2. **Anomaly-Based Detection for Novel Threats:** To detect and mitigate attacks that do not match any known signature.
3. **Behavioral Analysis for Persistent, Low-Profile Attacks:** Finally, behavioral-based detection catches complex attacks that mimic legitimate behavior.

This layered approach is particularly valuable as it allows organizations to adapt to evolving threats while minimizing false positives and ensuring that legitimate traffic is not interrupted. Together, these methods form a comprehensive, resilient, and highly effective DDoS defense framework.

Tools Used for the analysis:

In the code, various Python libraries for data analysis, machine learning, and visualization are utilized. Below is a breakdown of each tool and how it contributes to the code.

1. Pandas (pd)

   Purpose: Pandas is widely used for data handling and transformation, offering structures like DataFrames that organize data in tabular form. Usage: In this code, pd.read_csv() loads data from a CSV file into a DataFrame, making it easier to manipulate, filter, and clean the dataset. Later, pd.DataFrame() organizes packet information extracted from a .pcap file.

2. NumPy (np)

Purpose: NumPy facilitates the handling of large numerical data sets through arrays and mathematical operations. Usage: np.inf and np.nan handle infinite and missing values, while np.array() creates a sample array for testing with the classifier, allowing for flexible numerical computations.

3. Scikit-Learn (sklearn)

Purpose: This library offers a range of machine learning tools, including algorithms for classification, regression, clustering, and anomaly detection. Modules Used: Isolation Forest: An algorithm for identifying anomalies in a dataset, Isolation Forest isolates observations in the feature space using random decision trees. Each data point is either classified as an inlier or an outlier based on its isolation score, making it effective for detecting anomalies in network traffic data.

Random Forest Classifier: A versatile and powerful classification model based on an ensemble of decision trees. Each tree votes on the classification of data points, and the final prediction is determined by the majority vote, making it effective in classifying complex datasets and robust against overfitting.

StandardScaler: Scales features to have a mean of 0 and standard deviation of 1, aiding algorithms sensitive to feature scaling.

Usage:

Anomaly Detection (Isolation Forest): IsolationForest is used to detect unusual network activity in the scaled data. Its predict() method classifies entries as either anomalies or normal data points, allowing for the identification of potential intrusions.

Behavioural Analysis (Random Forest): RandomForestClassifier is trained on parsed network data to classify it as either normal or indicative of a DDoS attack. This model's performance is assessed using classification_report(), and train_test_split() divides the data into training and testing sets for accuracy evaluation.

4. Scapy (scapy)

Purpose: Scapy is used for network packet handling, popular in cybersecurity for analyzing and parsing network traffic. Usage: The rdpcap() function reads packet data from a .pcap file. For packets with an IP layer, data such as packet size and timing is extracted to create a dataset.

5. Matplotlib (plt)

Purpose: Matplotlib provides tools to create various types of visualizations. Usage: Scatter Plot: Highlights anomalies in network data by plotting Flow_IAT_Min over time. Histograms and Line Plots: Visualize distributions and time-based trends in the dataset, helping to identify potential anomalies.

6. Seaborn (sns)

Purpose: A visualization library based on Matplotlib, Seaborn creates visually appealing plots that help show data trends and distributions. Usage: Histplot: Displays packet size and duration distributions, with KDE for a smoother visualization. Line Plot: Tracks packet rate changes over time, identifying spikes or irregular patterns.

Code Workflow Summary

1. Data Loading and Cleaning: The CSV file is loaded and preprocessed, removing infinite values.
2. Anomaly Detection Using IsolationForest: An IsolationForest model identifies outliers, potentially representing unusual network activity.
3. PCAP File Parsing Using Scapy: Network packets are analyzed to extract relevant data.

4. Traffic Classification with RandomForest: The model is trained to classify traffic, assessing its accuracy afterward.

5. Visualization: Data insights are visualized, making it easier to detect anomalies and observe traffic trends.

This blend of tools forms a comprehensive analysis pipeline from data preprocessing to anomaly detection and classification, enhanced with visual insights.

ANOMALY-BASED DETECTION IN DDOS ATTACK:

STEPS INVOLVED:

- **Baseline Establishment**: Define normal network behaviour using historical data to establish a baseline for typical traffic patterns.

- **Traffic Monitoring**: Continuously monitor real-time traffic, analysing packets, flow rates, and user behaviour.

- **Anomaly Detection**: Compare current traffic against the baseline, using statistical or machine learning techniques to identify significant deviations.

- **Classification**: Filter out false positives and classify anomalies as potential attack patterns or non-attacks.

- **Alert Generation**: Generate alerts for potential attacks, notifying administrators or triggering automated responses.

- **Response and Mitigation**: Execute automated or manual responses, such as blocking malicious IPs or limiting traffic rates.

- **Post-Event Analysis**: Analyse the event to refine future detection accuracy and improve the baseline.

STEP BY STEP IMPLEMENTATION:

Step 1: Importing Libraries

D: > VIT- STUDY MATERIAL > project_cybersec > anomaly.py > ...

```
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

1. **Pandas**: For data manipulation and analysis.
2. **NumPy**: For numerical computations.
3. **IsolationForest**: Anomaly detection model from `sklearn.ensemble`.
4. **StandardScaler**: For feature scaling from `sklearn.preprocessing`.
5. **Matplotlib**: For visualizing the anomalies.

Step 2: Loading and Indexing the Data

```python
df = pd.read_csv(r'D:/VIT- STUDY MATERIAL/project_cybersec/network_traffic.csv')

index = pd.Index(range(2,len(df)+2))
df.index = index
```

1. **Load the CSV**: The code loads a network traffic dataset.
2. **Custom Index**: Sets a custom index starting from 2.

Step 3: Selecting Features for Anomaly Detection

```python
features = ['Flow_IAT_Min', 'Tot_Fwd_Pkts', 'Init_Bwd_Win_Bytes', 'Src_port']
X = df[features].copy()
```

- The features selected for anomaly detection are stored in a list, and the relevant columns are copied into `X` for further processing.

Step 4: Handling Missing Values and Infinite Values

```python
X.replace([np.inf, -np.inf], np.nan, inplace=True)
X.dropna(inplace=True)
```

- **Replace Infinite Values**: Any infinite values are replaced with NaN.
- **Remove NaNs**: Rows containing NaNs are dropped to ensure clean data for training the model.

Step 5: Scaling the Data

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- The data is scaled to have zero mean and unit variance, making the model training more effective and reducing bias from feature magnitude differences.

Step 6: Training the Isolation Forest Model

```python
isoforest = IsolationForest(n_estimators=100, contamination=0.01, random_state=42)
isoforest.fit(X_scaled)
```

1. **Create Model**: An Isolation Forest model is created with:
   - `n_estimators=100`: 100 base estimators (trees).
   - `contamination=0.01`: Assuming 1% of data points are anomalies.
   - `random_state=42`: For reproducibility.
2. **Fit Model**: The model is trained on the scaled data to learn patterns and detect anomalies.

Step 7: Predicting Anomalies

```python
predictions = isoforest.predict(X_scaled)
df['anomaly'] = np.nan
df.loc[X.index, 'anomaly'] = predictions
```

1. **Predict**: `isoforest.predict()` returns `1` for normal points and `-1` for anomalies.

2. **Store Predictions**: A new column, `anomaly`, is created to store predictions. The predictions for the selected features' rows are assigned to the corresponding indices.

Step 8: Isolating Anomalies and Normal Data

```
anomalies_df = df[df['Label'] == 1]
```

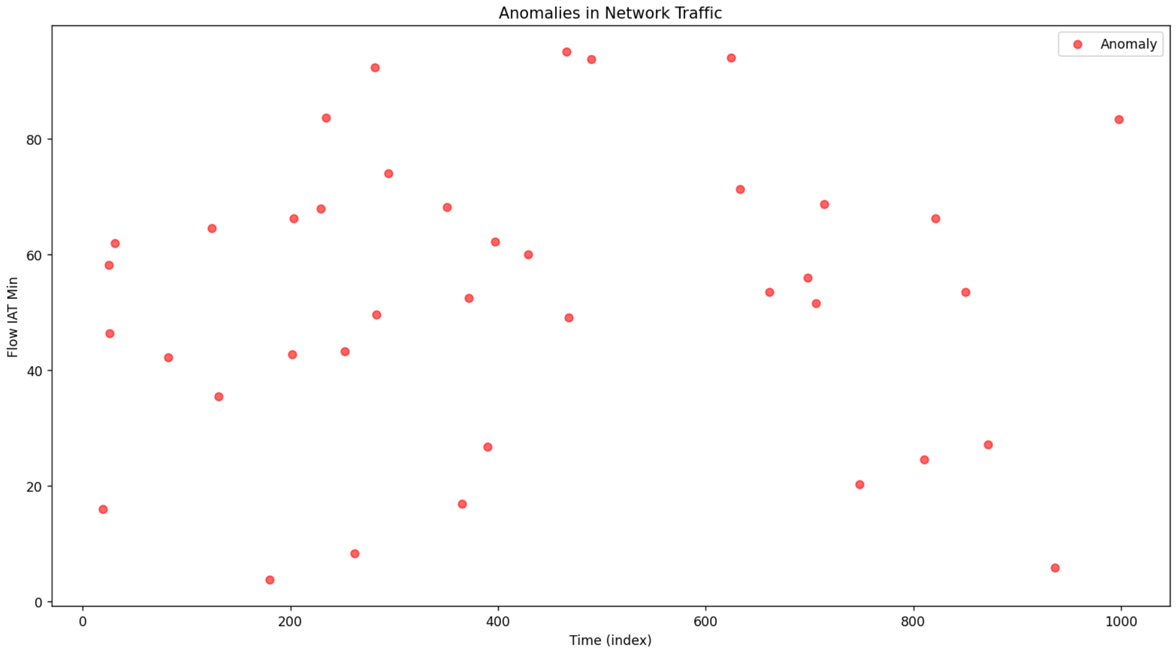
- `anomalies_df` is created by filtering rows where the `Label` column indicates an anomaly (assumed to be `1`).

Step 9: Visualization of Anomalies

```
print(anomalies_df)

normal_data = df[df['Label'] == 0]
anomalies_data = df[df['Label'] == 1]

plt.figure(figsize=(12, 6))
plt.scatter(anomalies_data.index, anomalies_data['Flow_IAT_Min'], color='red', label='Anomaly', alpha=0.6)

plt.title('Anomalies in Network Traffic')
plt.xlabel('Time (index)')
plt.ylabel('Flow IAT Min')
plt.legend()
plt.show()
```

1. **Separate Data**: `normal_data` and `anomalies_data` separate the data into normal and anomalous records.
2. **Plot**: A scatter plot shows anomalies over time using the `Flow_IAT_Min` feature to visualize when anomalies occur.

<u>OUTPUT SCREENSHOT:</u>



```
PS C:\Users\Pramiti> & C:/Users/Pramiti/AppData/Local/Programs/Python/Python311/python.exe "d:/VIT- STUDY MATERIAL/project_cybersec/anomaly.py"
            Src_IP      Dest_IP  Flow_IAT_Min  Src_port  Tot_Fwd_Pkts  Init_Bwd_Win_Bytes  Label  anomaly
19   192.168.1.173   10.0.0.27     16.022355     38686            45                2071      1      1.0
25    192.168.1.12  10.0.0.117     58.279506     40230            75                9711      1      1.0
26    192.168.1.30    10.0.0.9     46.437936     14679            63                8324      1      1.0
31    192.168.1.25   10.0.0.49     61.952729     33260            84                 828      1      1.0
82   192.168.1.115  10.0.0.144     42.249859     41745            20                9391      1      1.0
124  192.168.1.209  10.0.0.213     64.548832     52714            10                6055      1      1.0
131    192.168.1.4  10.0.0.146     35.450161      4239            95                6797      1      1.0
180  192.168.1.215   10.0.0.53      3.869744     47509            29                9585      1      1.0
202  192.168.1.154   10.0.0.70     42.702804      2280            68                2584      1      1.0
203   192.168.1.79  10.0.0.208     66.312755     37094            58                6809      1      1.0
229   192.168.1.72  10.0.0.209     67.894447     48445            74                8642      1      1.0
234  192.168.1.117  10.0.0.144     83.666680      8634            41                4577      1      1.0
252   192.168.1.93  10.0.0.186     43.258780     34747            56                2218      1      1.0
262  192.168.1.223   10.0.0.64      8.417782     11648            25                3254      1      1.0
281   192.168.1.67  10.0.0.177     92.423725      4112            56                6301      1      1.0
283   192.168.1.72  10.0.0.187     49.638597      8223            51                3795      1      1.0
294  192.168.1.102   10.0.0.53     74.021132     36493            23                2442      1      1.0
351  192.168.1.206   10.0.0.55     68.157765      3306            88                5701      1      1.0
365   192.168.1.53  10.0.0.229     16.875937      9027            26                5491      1      1.0
372   192.168.1.48  10.0.0.189     52.477187     12627            91                4080      1      1.0
390   192.168.1.24   10.0.0.29     26.820741     57376            46                6635      1      1.0
397  192.168.1.136  10.0.0.221     62.249530     57243            33                5238      1      1.0
429  192.168.1.145  10.0.0.162     60.062427     40962            92                5139      1      1.0
466  192.168.1.128   10.0.0.26     95.031345     35186            95                8237      1      1.0
468   192.168.1.95  10.0.0.146     49.064204     46608            27                5314      1      1.0
490  192.168.1.216   10.0.0.38     93.776519     24326            18                8341      1      1.0
624   192.168.1.88  10.0.0.184     94.108734     17259            61                2407      1      1.0
633   192.168.1.19   10.0.0.75     71.378828      1110            12                1201      1      1.0
661  192.168.1.186  10.0.0.201     53.497939     18840            52                9683      1      1.0
698  192.168.1.149   10.0.0.41     56.020428     44737            13                1467      1      1.0
706  192.168.1.242  10.0.0.121     51.565446     63216            26                9033      1      1.0
714  192.168.1.161  10.0.0.101     68.779021     34560            34                7119      1      1.0
748  192.168.1.155  10.0.0.175     20.366134     18574            82                3476      1      1.0
810   192.168.1.44  10.0.0.148     24.616613     23769            14                1533      1      1.0
821   192.168.1.98   10.0.0.66     66.237373      5506            83                4365      1      1.0
850  192.168.1.155  10.0.0.158     53.536200     57198            25                7539      1      1.0
872   192.168.1.12   10.0.0.77     27.215772     33043            82                8769      1      1.0
936   192.168.1.40  10.0.0.154      5.918341      1091            48                1268      1      1.0
998    192.168.1.6  10.0.0.235     83.360274     10755             8                 381      1      1.0
PS C:\Users\Pramiti>
```

VERIFICATION FROM CSV FILE:

1	Src_IP	Dest_IP	Flow_IAT_Min	Src_port	Tot_Fwd_Pkts	Init_Bwd_Win_E	Label
19	192.168.1.173	10.0.0.27	16.02235533	38686	45	2071	1
25	192.168.1.12	10.0.0.117	58.27950577	40230	75	9711	1
26	192.168.1.30	10.0.0.9	46.43793613	14679	63	8324	1
31	192.168.1.25	10.0.0.49	61.95272933	33260	84	828	1
82	192.168.1.115	10.0.0.144	42.24985917	41745	20	9391	1
124	192.168.1.209	10.0.0.213	64.54883232	52714	10	6055	1
131	192.168.1.4	10.0.0.146	35.4501614	4239	95	6797	1
180	192.168.1.215	10.0.0.53	3.86974449	47509	29	9585	1
202	192.168.1.154	10.0.0.70	42.70280432	2280	68	2584	1
203	192.168.1.79	10.0.0.208	66.31275474	37094	58	6809	1
229	192.168.1.72	10.0.0.209	67.89444682	48445	74	8642	1
234	192.168.1.117	10.0.0.144	83.66667974	8634	41	4577	1
252	192.168.1.93	10.0.0.186	43.25878031	34747	56	2218	1
262	192.168.1.223	10.0.0.64	8.417782495	11648	25	3254	1
281	192.168.1.67	10.0.0.177	92.42372519	4112	56	6301	1
283	192.168.1.72	10.0.0.187	49.63859699	8223	51	3795	1
294	192.168.1.102	10.0.0.53	74.02113168	36493	23	2442	1
351	192.168.1.206	10.0.0.55	68.15776526	3306	88	5701	1
365	192.168.1.53	10.0.0.229	16.87593743	9027	26	5491	1
372	192.168.1.48	10.0.0.189	52.47718747	12627	91	4080	1
390	192.168.1.24	10.0.0.29	26.82074101	57376	46	6635	1
397	192.168.1.136	10.0.0.221	62.24952953	57243	33	5238	1
429	192.168.1.145	10.0.0.162	60.06242713	40962	92	5139	1
466	192.168.1.128	10.0.0.26	95.03134468	35186	95	8237	1
468	192.168.1.95	10.0.0.146	49.06420364	46608	27	5314	1
490	192.168.1.216	10.0.0.38	93.77651852	24326	18	8341	1
624	192.168.1.88	10.0.0.184	94.10873356	17259	61	2407	1
633	192.168.1.19	10.0.0.75	71.37882824	1110	12	1201	1
661	192.168.1.186	10.0.0.201	53.49793919	18840	52	9683	1
698	192.168.1.149	10.0.0.41	56.02042828	44737	13	1467	1

network_traffic

Ready 39 of 1000 records found Accessibility: Unavailable

Behavioural Analysis:

Step 1: Importing Libraries

```
import numpy as np
import pandas as pd
from scapy.all import rdpcap
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from matplotlib import pyplot as plt
import seaborn as sns
```

- **numpy and pandas:** Libraries for handling arrays and dataframes, respectively, which are essential for data manipulation and preparation.
- **rdpcap from scapy:** Reads packet capture files (.pcap), which contain network traffic data.
- **RandomForestClassifier from sklearn:** A machine learning model for classifying network traffic as normal or potentially indicative of a DDoS attack.
- **Train_test_split:** Divides the dataset into training and testing sets to evaluate model performance.
- **Classification_report:** Generates a summary of the model's performance metrics (e.g., precision, recall, F1-score).
- **Matplotlib.pyplot and seaborn:** Libraries for creating visualizations, allowing us to analyze distributions and trends.

Step 2: Loading and Indexing the Data

```
df = parse_pcap(r"C:\Users\rohan\Downloads\Project\Project\data\pcap_files\traffic.pcap")
```

- **Load the PCAP**: The code loads a network traffic dataset.

Step 3: Prepare Data for Model Training

```
X = df[['packet_size', 'packet_rate', 'duration']]
y = df['is_ddos']
```

- X contains the features: packet_size, packet_rate, and duration.
- y contains the target variable (is_ddos), initially set to 0 for normal traffic.

Step 4: Split the Dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

- Split the dataset into training and testing sets. The test set takes 30% of the data.

Step 5: Train the Random Forest Classifier

```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

- Initialize and train the RandomForestClassifier on the training data

Step 6: Make Predictions

```
y_pred = clf.predict(X_test)
```

- Use the trained classifier to predict values for X_test.

Step 7: Evaluate the Model

```
print(classification_report(y_test, y_pred))
```

- Generate a classification report to see metrics like precision, recall, and F1-score, which assess model performance.

Step 8: Predict a New Sample

```
new_sample = np.array([[500, 300, 50]])
prediction = clf.predict(new_sample)
print(f"Prediction for the new sample: {'DDoS' if prediction[0] == 1 else 'Normal'}")
```

- Create a new sample (with packet size 500, rate 300, and duration 50) and use the trained model to classify it.

Step 9: Visualization of Anomalies

Use Matplotlib and Seaborn to plot distributions and trends in the dataset.

- Distribution of Packet Sizes

```
plt.figure(figsize=(10, 6))
sns.histplot(df['packet_size'], bins=50, kde=True)
plt.title('Distribution of Packet Sizes')
plt.xlabel('Packet Size')
plt.ylabel('Frequency')
plt.show()
```

- **Packet Rate Over time**

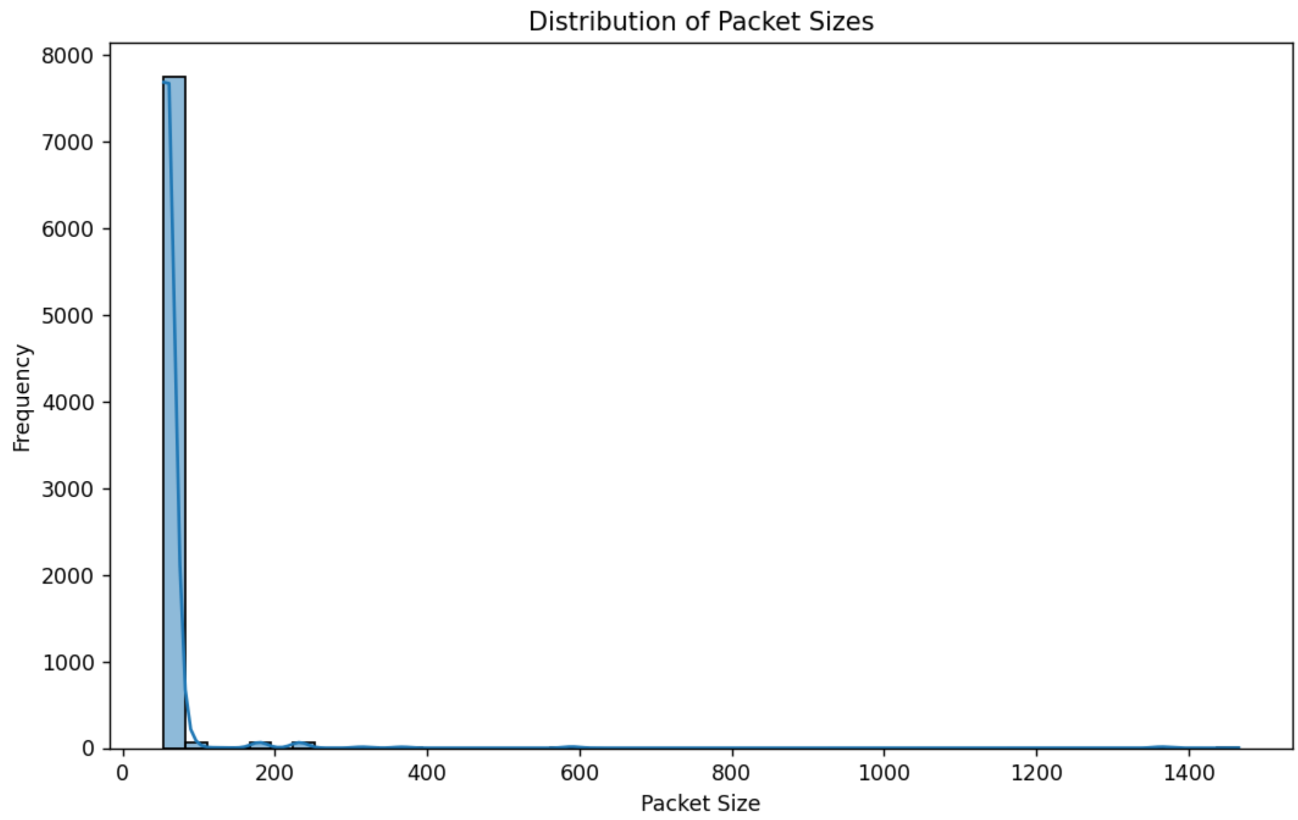
```python
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x=df.index, y='packet_rate')
plt.title('Packet Rate Over Time')
plt.xlabel('Time')
plt.ylabel('Packet Rate')
plt.show()
```

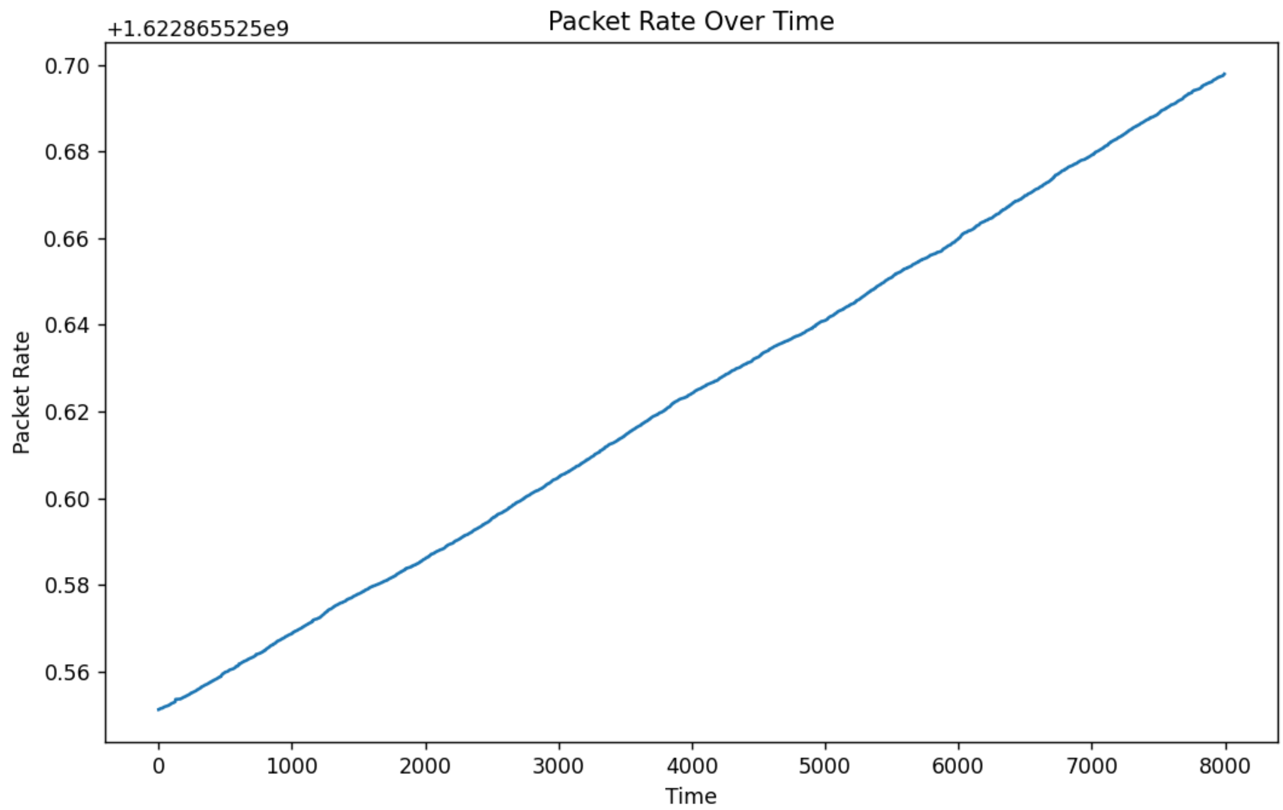
- **Duration of Packets**

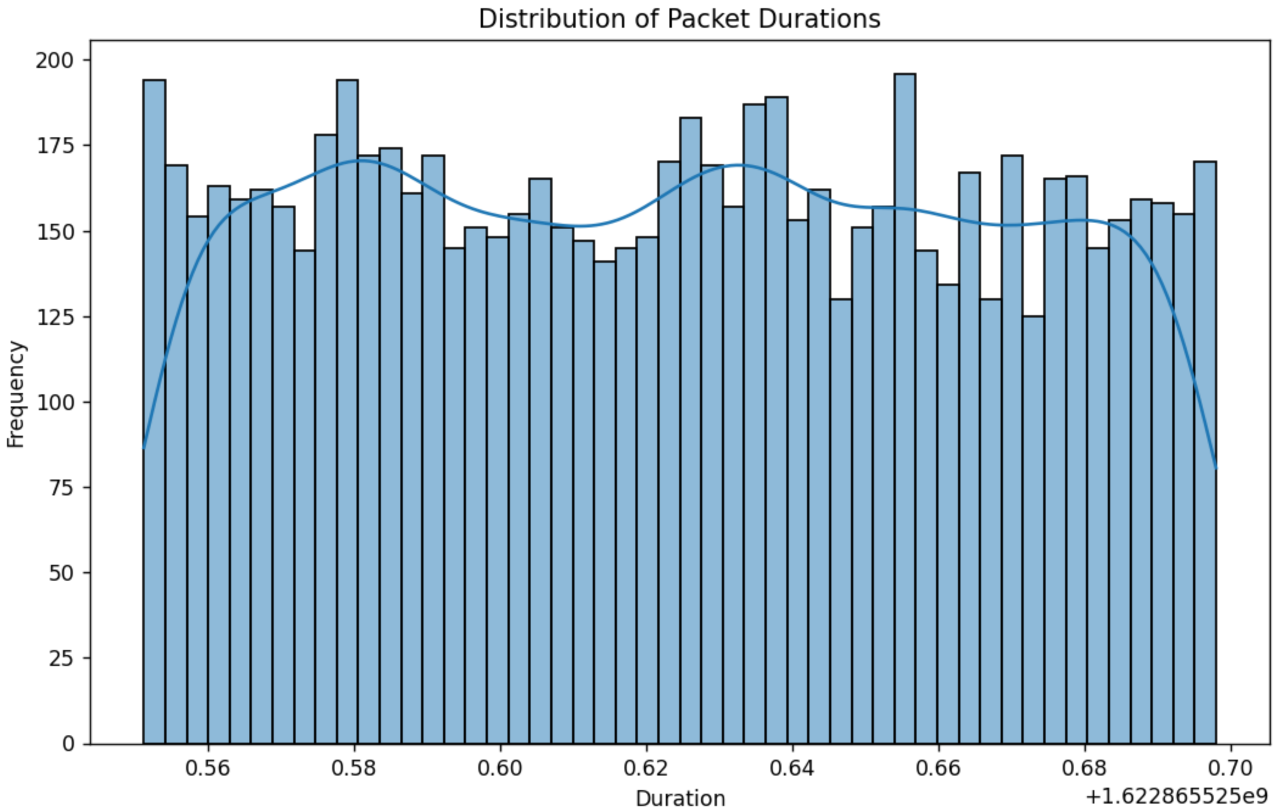
```python
# Duration of Packets
plt.figure(figsize=(10, 6))
sns.histplot(df['duration'], bins=50, kde=True)
plt.title('Distribution of Packet Durations')
plt.xlabel('Duration')
plt.ylabel('Frequency')
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2399

    accuracy                           1.00      2399
   macro avg       1.00      1.00      1.00      2399
weighted avg       1.00      1.00      1.00      2399

C:\Python38\lib\site-packages\sklearn\base.py:465: UserWarning: X does not have valid feature names, but RandomForestClassi
fier was fitted with feature names
  warnings.warn(
Prediction for the new sample: Normal
```

<u>OUTPUT SCREENSHOT:</u>


Distribution of Packet Sizes
Frequency
Packet Size
0
1000
2000
3000
4000
5000
6000
7000
8000
0
200
400
600
800
1000
1200
1400


Packet Rate Over Time
+1.622865525e9
Packet Rate
Time
0.56
0.58
0.60
0.62
0.64
0.66
0.68
0.70
0
1000
2000
3000
4000
5000
6000
7000
8000



Signature based:

Step 1: Importing Libraries

- **json**: In signature_matching.py, the json library is used to load and parse a JSON file containing DDoS attack signatures. This allows easy access to predefined patterns that can be matched against extracted network features, aiding in DDoS attack identification.
- **dpkt**: The dpkt library in data_collection.py and feature_extraction.py is crucial for reading and parsing network packet data from .pcap files. It enables us to work with the binary structure of packets, extract packet details, and perform network analysis.
- **socket**: This library, used in both data_collection.py and feature_extraction.py, provides methods to convert binary IP addresses to human-readable form. This makes it easier to debug and interpret IP addresses during analysis.

src > data_collection.py > read_pcap_file

```
import dpkt
import socket
```

src > feature_extraction.py > extract_features

```
import dpkt
import socket
```

```
src > signature_matching.py > match_signature
import json
```

Step 2: Loading and Indexing the Data

- **Load the PCAP**: In data_collection.py, the read_pcap_file function is responsible for loading the PCAP file (specified as 'data/pcap_files/traffic.pcap' in main.py). The function opens the file in binary mode and iterates over each packet using dpkt.pcap.Reader. It extracts essential packet information by:
    - Checking if the packet is an Ethernet frame and contains IP data.
    - Filtering for TCP packets.
    - Extracting the source and destination IP addresses and ports using socket.inet_ntoa() for

```
def read_pcap_file(pcap_file):
    with open(pcap_file, 'rb') as f:
        pcap = dpkt.pcap.Reader(f)
        for ts, buf in pcap:
            eth = dpkt.ethernet.Ethernet(buf)
            if eth.type != dpkt.ethernet.ETH_TYPE_IP:
                continue
            ip = eth.data
            if ip.p == dpkt.ip.IP_PROTO_TCP:
                tcp = ip.data
                # Extract features for analysis
                source_ip = socket.inet_ntoa(ip.src)
                destination_ip = socket.inet_ntoa(ip.dst)
                source_port = tcp.sport
                destination_port = tcp.dport
                # Perform feature-based analysis and signature matching
                # ...
                yield {
                    'source_ip': source_ip,
                    'destination_ip': destination_ip,
                    'source_port': source_port,
                    'destination_port': destination_port,
                    # Add other extracted features as needed
                }
```

better readability.

Step 3: Prepare Data for Analysis

- **Extract Features**: In feature_extraction.py, the extract_features function reads each packet from the specified PCAP file and extracts the following features: source_ip, destination_ip, source_port, and destination_port. This data is stored in a list, features, for further processing.
    - Each feature is appended to the features list as a dictionary, where the extracted IPs and ports are recorded.
    - This extracted data is later used for matching against known attack signatures in `signature_matching.py`.

```
def extract_features(pcap_file):
    features = []
    with open(pcap_file, 'rb') as f:
        pcap = dpkt.pcap.Reader(f)
        for ts, buf in pcap:
            eth = dpkt.ethernet.Ethernet(buf)
            if eth.type != dpkt.ethernet.ETH_TYPE_IP:
                continue
            ip = eth.data
            if ip.p == dpkt.ip.IP_PROTO_TCP:
                tcp = ip.data
                features.append({
                    'source_ip': socket.inet_ntoa(ip.src),
                    'destination_ip': socket.inet_ntoa(ip.dst),
                    'source_port': tcp.sport,
                    'destination_port': tcp.dport,
                    # Add other relevant features as needed
                })
    return features
```

Step 4: Signature Matching

- **Load Signatures**: In signature_matching.py, the match_signature function reads known DDoS attack signatures from the JSON file, data/signatures.json. This file contains signature patterns of malicious IP addresses to be compared with the extracted features.
- **Match Network Features**: The function iterates over each feature and compares it with every signature. If any feature matches a signature (i.e., source and destination IPs align with known attack patterns), it returns True, indicating that a potential DDoS attack has been detected.
    - If no matches are found, the function returns False, signaling no known threat has been detected.

```
def match_signature(features, signatures):
    with open('data/signatures.json', 'r') as f:
        signatures = json.load(f)

    for feature in features:
        for signature in signatures:
            if (
                feature['source_ip'] == signature['source_ip'] and
                feature['destination_ip'] == signature['destination_ip']):
                return True
    return False
```

Step 5: Generate Alerts

- **Alert System**: In alert_generation.py, the generate_alert function is invoked when no matching signatures are found in match_signature. This function receives the source_ip and destination_ip as arguments, which represent the IPs involved in the suspicious traffic. It prints a message that signals a potential DDoS attack:
    - Example output: DDoS attack detected from <source_ip> to <destination_ip>.
    - This output can be extended to log the alert details to a file or trigger an alert notification system.

```
def generate_alert(source_ip, destination_ip):
    print(f"DDoS attack detected from {source_ip} to {destination_ip}")
    # Log the alert or send a notification
```

Step 6: Running the Main Program

- **Main Execution**: In main.py, the main program starts by specifying the path to the PCAP file and invoking the read_pcap_file function to load the traffic data.
    - The extract_features function is called to retrieve features from each packet.
    - The code iterates over each feature and checks it against known signatures. If a DDoS pattern is not detected, an alert is generated using generate_alert.
- **Timing Control**: The time.sleep(1.5) line introduces a 1.5-second delay between processing each packet. This simulates a real-time traffic analysis environment, allowing for efficient and manageable alert generation.

```python
from data_collection import read_pcap_file
from feature_extraction import extract_features
from signature_matching import match_signature
from alert_generation import generate_alert
import time

if __name__ == '__main__':
    # Adjust interface and capture rate as needed
    pcap_file = 'data/pcap_files/traffic.pcap'
    read_pcap_file(pcap_file)
    features = extract_features(pcap_file)

    for i in pcap_file:
      if match_signature(features, signatures='data/signatures.json') is False:
          generate_alert(features[0]['source_ip'], features[0]['destination_ip'])
      time.sleep(1.5)
```

RECENT RESEARCH ON DDoS ATTACKS:

Recent research into Distributed Denial of Service (DDoS) attack detection has seen significant improvements in the use of machine learning, hybrid methods, and intelligent agent-based systems. Here's a summary of these advancements, including statistics from recent studies:

Signature-based Detection: Signature-based systems are widely used to detect known attack patterns and remain one of the most efficient methods for such tasks. A study by Javed et al. (2023) demonstrated that combining signature-based detection with machine learning techniques achieved 99.8% accuracy in detecting known DDoS attacks. However, this method faces challenges when dealing with novel or previously unseen attacks, where detection performance can drop significantly. To address this, another hybrid model combining signature-based and anomaly-based detection reported 99% detection accuracy for known attack signatures, with a low false positive rate of 2-4%

Anomaly-based Detection: Anomaly detection focuses on identifying traffic that deviates from established patterns, enabling it to detect unknown or zero-day DDoS attacks. One study reported that a machine learning-enhanced anomaly detection system achieved a 96.5% detection accuracy, but struggled with a relatively high 13% false positive rate. To improve this, a more advanced deep learning approach using attention mechanisms reached 98% detection accuracy, significantly reducing false positives and improving the overall performance of the system.

Behavioral-based Detection: Behavioral-based methods focus on long-term analysis of network activity to identify subtle behavioral anomalies that could indicate an ongoing DDoS attack. In a recent study, an intelligent agent-based system using automatic feature extraction and selection achieved a remarkable 99.7% detection accuracy for identifying DDoS attacks, outperforming traditional methods that typically fall in the 90-95% range. This highlights the effectiveness of behavioral-based detection when integrated with dynamic, intelligent agents that adapt to changing network conditions.

Hybrid Detection Models: Combining multiple detection methods into hybrid systems has been a promising approach to tackle both known and unknown attack patterns. A hybrid model that integrates signature-based and anomaly-based detection achieved 98.5% detection accuracy with a significantly reduced false positive rate of about 4%. This model dynamically adjusts to various attack scenarios, ensuring high detection accuracy and minimizing false alarms.

These studies underscore the ongoing advancements in DDoS detection, with a clear trend towards more sophisticated, hybrid systems that leverage machine learning to enhance both accuracy and adaptability. The improved detection rates (ranging from 96.5% to 99.8%) and reduced false positives (as low as 2-4%) demonstrate the progress being made in combating DDoS threats with advanced detection techniques.

GITHUB LINK:

https://github.com/pramitiss/cybersec_project.git

BIBLIOGRAPHY

Anomaly based:
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c02f4026b7e60ef2de5d2c337d9611d3c2bf6827

https://www.mdpi.com/

https://www.cisecurity.org/

https://www.bluegoatcyber.com/

Behavioural based:

https://ieeexplore.ieee.org/abstract/document/8102932

Signature based:

https://link.springer.com/chapter/10.1007/978-3-031-04036-8_6

http://www.ijpe-online.com/EN/article/downloadArticleFile.do?attachType=PDF&id=4703

https://ieeexplore.ieee.org/abstract/document/9511420/

https://www.researchgate.net/profile/Kumar-Virendra-2/publication/384141383_Real-time_signature-based_detection_and_prevention_of_DDOS_attacks_in_cloud_environments/links/66ebf11519c9496b1facbed2/Real-time-signature-based-detection-and-prevention-of-DDOS-attacks-in-cloud-environments.pdf

Research paper Link:

https://www.mdpi.com/2224-2708/12/4/51

https://www.ijcaonline.org/

https://www.mdpi.com/1424-8220/23/6/3333