INSTITUT SUPÉRIEUR D'ÉLECTRONIQUE DE PARIS (ISEP)

DETECTING SYN FLOODING ATTACKS USING THE SLIDING WINDOWS TECHNIQUE AND THE ISOLATION FOREST MACHINE LEARNING ALGORITHM

S/N	Student Name	Student Number	Major	Supervisor
01	LEIGHA Ifiyemi	62769	Digital Security and Networks	CHABCHOUB Yousra

February 2024

ABSTRACT

This research focuses on addressing the increasingly prevalent issue of SYN flooding attacks, a form of Denial of Service (DoS) attack, which poses significant threats to the stability and security of online services. The core objective is to detect such attacks using the Sliding Windows technique with a suite of machine learning algorithms: Isolation Forest Algorithm.

The methodology employed involves a systematic analysis of network traffic data, utilizing the Sliding Windows technique to dynamically segment this data for real-time analysis. This segmentation allows for the continuous evaluation of traffic patterns, facilitating the early detection of anomalies indicative of a SYN flooding attack. The research integrates and compares the effectiveness of the chosen machine learning algorithm in identifying these anomalies, and its efficacy measured against a set of performance metrics including accuracy, precision, recall, and F1 score.

Key findings demonstrate that the Isolation Forest algorithm excelled in scenarios with high data dimensionality.

In conclusion, the research establishes that the combination of the Sliding Windows technique with the Isolation Forest machine learning algorithm provides a viable and effective solution for detecting SYN flooding attacks. However, the study also underscores the necessity for algorithm-specific tuning to cater to different network environments and attack complexities. Future research directions include the refinement of the Y Algorithm and the exploration of hybrid models that combine the strengths of the individual algorithms for enhanced detection accuracy and efficiency.

ACKNOWLEDGEMENT

This project, a culmination of dedicated efforts and in-depth research, would not have been possible without the support and guidance of various individuals and resources. I extend my sincere gratitude to those who contributed to the successful completion of this study on "Detecting SYN Flooding Attacks Using the Sliding Windows Technique and the Isolation Forest Unsupervised Machine Learning Algorithm."

First and foremost, I express our deepest appreciation to my supervisor, Yousra Chabchoub, whose expertise and insightful guidance have been invaluable throughout this journey. Your unwavering support, constructive criticism, and encouragement have been pivotal in shaping our research and steering it towards fruition.

I am also immensely grateful to my fellow colleagues for their support in helping me working on this project. This work helped me forged a comprehensive and robust study.

Special thanks to the various authors and researchers whose papers and publications provided us with a rich source of knowledge and state-of-the-art methodologies in the field of network security and machine learning. These resources have been instrumental in broadening our understanding and shaping my approach to tackling the problem of SYN flooding attacks.

I would also like to extend my gratitude to my university, Institut Superieur d'Electronique De Paris [ISEP], for providing me with the necessary resources, facilities, and an environment conducive to research. The access to a conducive environment, computing resources, and a supportive academic community played a crucial role in our research process.

Lastly, I acknowledge my family and friends for their constant encouragement, patience, and understanding throughout the duration of this project. Your moral support and belief in our capabilities have been a source of motivation and strength.

In conclusion, this project is not just a reflection of our hard work but also a testament to the collective effort and support of everyone who contributed in various capacities. I am deeply grateful to all who have been part of this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

1.1 Overview of SYN Flooding Attacks

SYN flooding attacks, a prevalent form of Denial-of-Service (DoS), exploit the TCP handshake process to overwhelm target systems, leading to server unresponsiveness to legitimate traffic. This type of attack, which exploits the TCP handshake process by sending numerous SYN requests without completing the connection, results in the server allocating resources for these incomplete handshakes, leading to resource exhaustion [Tang, H., et al., 2009]. The incapacitation of web servers by these attacks makes them inaccessible to legitimate users, posing a substantial security concern [Sun, F., & Wu, Z., 2011].

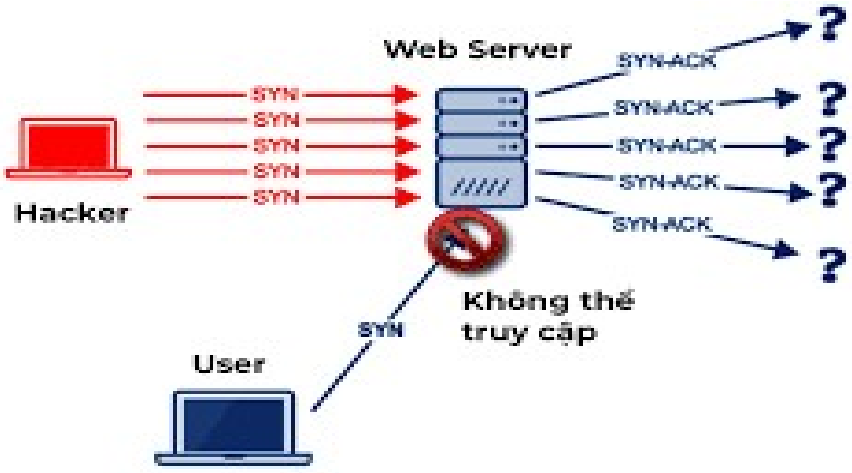


Figure 1: SYN Flooding Attack.

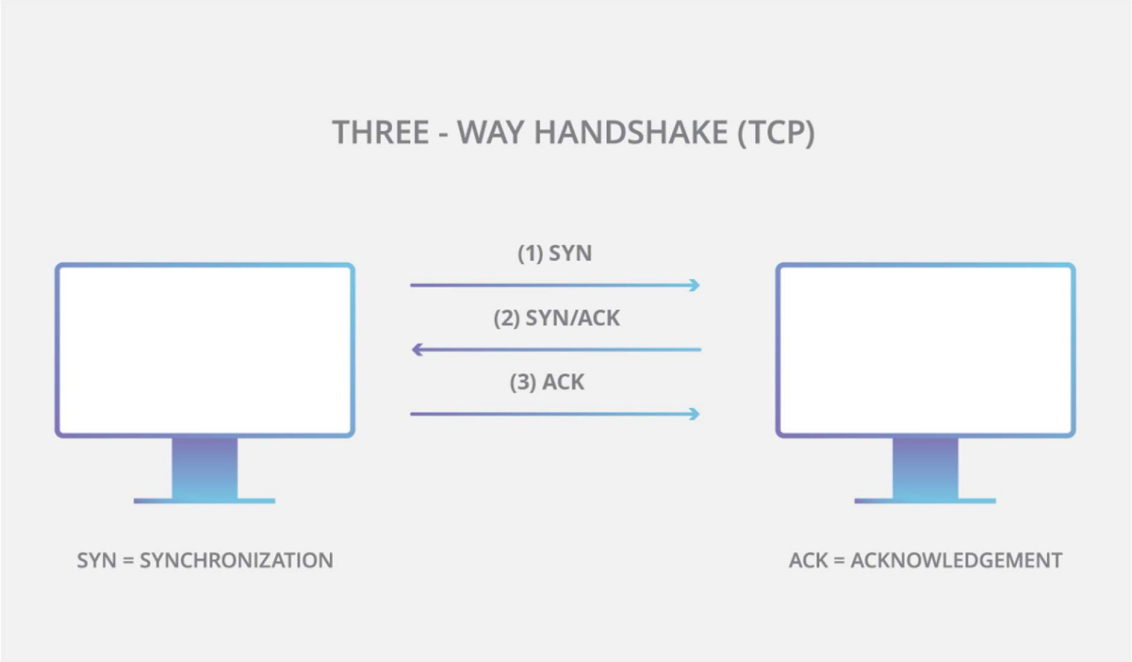


Figure 2: The Three-way handshake.

1.2 Problem Statement and Research Significance

Distinguishing between legitimate and malicious SYN packets presents a significant challenge in mitigating SYN flooding attacks [Tuncer, T., & Tatar, Y., 2008]. These attacks can disrupt online services, affecting both users and service providers. This research aims to develop effective detection strategies, enhancing network security by introducing reliable mechanisms for the identification and neutralization of SYN flooding attacks, ensuring the continuity of online services [Zouka, H. A. E., n.d.].

1.3 Research Objectives and Scope

The primary goal of this project is to create an effective approach for detecting SYN flooding attacks using a combination of the sliding window technique and advanced machine learning algorithms, including Isolation Forest, K-means, One-Class Support Vector Machine, and DBSCAN [Zhang, L., & Liu, L., 2022; Nan Haymarn Oo, A. Maw., 2019]. The scope involves developing, implementing, and testing these methods to provide a novel solution for this cybersecurity threat.

1.4 Methodological Approach

The methodological approach of this research involves the collection and analysis of network traffic data, employing the sliding window technique for real-time traffic pattern analysis [Chibani, N., et al., 2022]. The performance of each

machine learning algorithm in detecting SYN flooding attacks will be evaluated and compared based on accuracy, precision, and computational efficiency [Wagner, C., et al., 2011].

1.5 Contribution to the Field

This research is expected to contribute to the understanding of SYN flooding attack mechanisms and their impact on network security. It seeks to introduce effective tools and techniques for the detection and mitigation of these attacks, providing valuable insights and practical solutions to the field of cybersecurity [Campazas-Vega, A., et al., 2023].

Chapter 2
Literature Review

2.1 Overview of SYN Flooding Attacks

SYN flooding attacks, characterized by the abuse of the TCP handshake mechanism, pose a significant threat to network security. These attacks, which inundate a target server with SYN requests without completing the handshake, lead to the exhaustion of server resources and denial of service to legitimate users [Tang, H., et al., 2009]. Research by Sun and Wu [2011] highlights the impact of these attacks on web services, emphasizing the need for effective detection mechanisms. Studies such as those by Tuncer and Tatar [2008] and Zouka [n.d.] provide insights into the complexities of identifying and mitigating SYN flooding attacks.

2.2 Sliding Windows Technique

The sliding windows technique is a method used in network traffic analysis for real-time monitoring and anomaly detection. Its application in detecting SYN flooding attacks involves continuously analyzing segments of network traffic to identify unusual patterns indicative of an attack [Chibani, N., et al., 2022]. Nan Haymarn Oo and A. Maw [2019] illustrate the effectiveness of this technique in detecting SYN flooding in Software-Defined Networks (SDNs), highlighting its relevance and applicability.

2.3 Machine Learning Algorithms for SYN Flooding Detection

- **Isolation Forest:**
  The Isolation Forest algorithm, known for its efficiency in anomaly detection, has been applied to various cybersecurity challenges [Zhang, L., & Liu, L., 2022; Negi, K., et al., 2022]. Its use in network security, particularly in detecting anomalies such as SYN flooding attacks, demonstrates its potential in this field [S. S., S. G., & Priya, B., 2022].
- **K-means Clustering:**
  K-means, a widely used clustering algorithm, has been explored in the context of network anomaly detection, including DDoS attacks [Gu, Y., et al., 2017]. Pramana et al. [2015] discuss its application in DDoS detection, emphasizing its effectiveness in identifying unusual traffic patterns.
- **One-Class Support Vector Machine (OC-SVM):**
  The OC-SVM algorithm, known for its proficiency in detecting outliers, has been applied in network security [Yang, K., et al., 2021]. Santiago-Paz et al. [2015] highlight its use in anomaly detection, particularly in network traffic.
- **DBSCAN Algorithm:**
  As a density-based clustering algorithm, DBSCAN has been explored in various fields, including network security [Campazas-Vega, A., et al., 2023]. Its application in detecting network anomalies aligns with the objectives of this research.

- **Software-Defined Networking (SDN) Approaches:**
  Recent developments in SDN have led to innovative methods for detecting SYN flooding attacks. A 2023 study presented a novel approach combining cuckoo hashing and whitelist techniques within SDN, significantly improving traffic management and detection accuracy. This advancement showcases the potential of SDN in enhancing cybersecurity measures against such attacks [Meng-Hsun Tsai, et al., "Detection and Mitigation of SYN Flooding Attacks through SYN/ACK Packets and Black/White Lists," Sensors, vol. 23, no. 8, pp. 3817, April 2023, doi: 10.3390/s23083817].

2.4 Gaps in Current Research

While existing studies have explored various aspects of SYN flooding attacks and their detection, there remain gaps, particularly in the integration of multiple machine learning algorithms with real-time traffic analysis techniques like the sliding windows method. This research aims to bridge these gaps by conducting a comparative analysis of different algorithms' performance in SYN flooding detection and integrating them with the sliding windows technique for enhanced real-time detection capabilities. The study also seeks to contribute to the development of more adaptive, efficient, and robust detection systems capable of responding to evolving network threats.

Chapter 3
Methodology

3.1 Data Collection

- **Sources of Data:**
  The sample data is a TCPDUMP Network Logs collected from the network environment of Orange S.A. Telecommunications company in Paris, France.

- **Nature of Data:**
  The nature of the data is crucial for understanding network behavior and identifying anomalies. Here's how we would describe the nature of the data based on the given sample data and their column names:

  Start and End Times (date_debut_us, date_fin_s, date_fin_us):

  These columns indicate the start and end times of network traffic in microseconds and seconds, providing a temporal context for each data packet.

  Packet Information (nb_pkts, nb_syn, nb_synack, nb_fin, nb_rst): These columns include the count of various types of packets, such as total packets (nb_pkts), SYN packets (nb_syn), SYN-ACK packets (nb_synack), FIN packets (nb_fin), and RST packets (nb_rst). A high number of SYN packets relative to SYN-ACK packets could indicate a SYN flooding attack.

  Volume of Traffic (volume_total):

  This represents the total volume of data transmitted in the traffic, which can be useful for identifying large-scale anomalies in network traffic.

  Traffic Direction (sens_sur_le_lien):

  Indicates the direction of the traffic, whether incoming or outgoing ('in' or 'out'), which is important for understanding the source and target of potential attacks.

  By analyzing these parameters, particularly the ratio of SYN to SYN-ACK packets and the volume of traffic over time, I can identify patterns consistent with SYN flooding attacks. For instance, a surge in SYN packets without a corresponding increase in SYN-ACK packets within a short time frame might signal an ongoing attack. This analysis forms a critical part of my project's methodology in detecting SYN flooding anomalies in network traffic.

3.2 Algorithm Implementation

- **Isolation Forest:**
  - **Data Preparation and Preprocessing:**
    - The **read_data** function reads network traffic data from a file, using columns like 'Destination IP', 'Source IP', 'Source Port', 'Destination Port', 'Protocol', 'Start Timestamp', and 'SYN Flag'.
    - The **filter_syn_packets** function filters the dataset to include only packets with the SYN flag set. This is crucial for identifying potential SYN flooding attacks.
    - The **convert_to_datetime** function converts 'Start Timestamp' to a datetime format for easier analysis.
  - **Isolation Forest Application:**
    - The **detect_anomalies_with_isolation_forest** function is the core of the algorithm. It first groups the data by time intervals, destination IP, and destination port, then counts the number of SYN packets in each group.
    - The Isolation Forest model is applied to this grouped data. The model uses **contamination=0.0005**, which specifies the expected proportion of outliers in the data.
    - The model then fits to the 'Count' of SYN packets and predicts anomalies, marking unusual surges in SYN packet counts as potential SYN flooding activities.
  - **Visualization and Model Evaluation:**
    - The code includes functionality for visualizing the detected anomalies and calculating response times for significant bursts of SYN packets.
    - The model's effectiveness is evaluated using standard metrics like accuracy, precision, recall, and F1 score. These metrics provide insights into how well the model can detect SYN flooding attacks.
  - **Resource Management and Performance Metrics:**
    - The code measures execution time, CPU usage, and memory usage, indicating the efficiency and resource demands of the algorithm.
    - This is crucial for understanding the feasibility of deploying this solution in real-time network environments.
  - This implementation of the Isolation Forest algorithm is designed to detect potential SYN flooding attacks by analyzing network traffic data. The approach focuses on identifying anomalies in SYN packet counts within specific time intervals, which is a key indicator of SYN flooding. The algorithm's application, combined with visualization and performance evaluation, forms an integral part of the project's methodology for detecting SYN flooding anomalies.
  -

3.3 Sliding Windows Technique Application

3.3.1 What is Actually the Sliding Window Technique

The sliding window (windowing) technique is used by Transmission Control Protocol (TCP) to manage the flow of packets between two computers or network hosts. TCP is a core component of the Internet Protocol suite and operates at the transport layer.

The data link layer in the Open Systems Interconnection model also supports a class of protocols that use the sliding window technique. Protocols that support this technique are often referred to *sliding window protocols*. Sliding window protocols ensure the reliable and sequential delivery of data packets between network devices.

3.3.2 How Does it Work?

A sliding window protocol controls and optimizes packet flow between a sender and receiver, while ensuring a balanced approach to packet delivery. The protocol requires the receiver to acknowledge receipt of each data packet, and it enables the receiver to use a single acknowledgment (ACK) to confirm the delivery of multiple packets.

The receiver maintains a buffer to manage the flow of data packets. The receive buffer holds the packets that have been sent by the sender but have not yet been processed. During data transmission, the receiver notifies the sender of the amount of free space available in the receiver buffer. This space is referred to as the *receive window,* which is the buffer size less the amount of unprocessed data. The sender cannot send more data packets than the amount of space available in the receive window.

Data packets are numbered sequentially so they can be tracked when data is being transmitted from the sender to the receiver. During the transmission process, the data packets pass through one of four stages:

1. Sent and acknowledged by the receiver.
2. Sent but not acknowledged by the receiver.
3. Not sent but the receiver is ready to accept them.
4. Not sent and the receiver is not ready to accept them.

Figure 2 provides a conceptual overview of the packets in a data segment as they pass through each stage. The packets are numbered sequentially, which serves as a reference point for both the sender and receiver.

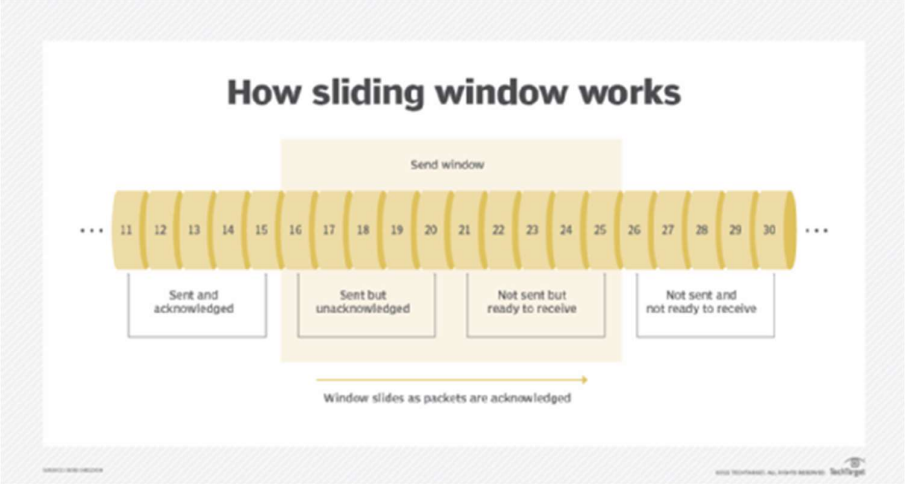


Figure 3. Conceptual overview of the packets in a data segment as they pass through each sliding window stage

Under normal operations, each data packet passes through each stage in the order specified. The sender supports its own send window that corresponds to the receive window. The send window includes the data packets in the second and third stages.

For example, Figure 2 shows that packets 16-20 have been sent, but the receiver has not yet acknowledged them. Figure 2 also shows that the receiver is ready to accept packets 21-25, but they have not yet been sent. In all, the send window includes packets 16-25. All packets up to 15 have been sent and acknowledged, and packets 26 onward have not been sent nor are they ready to be accepted.

As data is transmitted, the receiver acknowledges the packets it has accepted and continues to provide an updated status of the size of receive window. The sender adjusts its send window accordingly to reflect which sent packets have yet to be acknowledged and which additional packets can now be sent.

With each ACK and update on the receive window size, the sender adjusts the window to incorporate one or more of the next consecutive data packets. In effect, the send window slides along the data segment -- from left to right in Figure 2 -- to incorporate the next set of data packets in the sequence, continuously responding to the receiver's acknowledgments and receive window updates.

Each ACK sent by the receiver acknowledges packet receipt and specifies the current size of the receive window. A single ACK can apply to one or more packets. For example, returning to Figure 2, if the receiver sends an ACK for packet 20, all packets up to and including 20 have been successfully received.

However, if the receiver acknowledges only packet 18, it means that the receiver has accepted all packets up to and including 18 but has still not received packets 19 and 20. If the sender does not receive an ACK for these packets within a specified amount of time, the sender will resend the two packets.

The sender must continuously adjust its send window based on received ACKs and amount of space available in the receive window. If the window size is zero, the sender must wait for an ACK before sending the next chunk of data. If the receiver reports that the receive window size is larger than the size of a single data packet, the sender can send multiple packets without waiting for an ACK.

Transmitting multiple packets between ACKs enables the data to be transferred faster and more efficiently than if an ACK must be received prior to each transmission.

3.3.3 How Does the Sliding Window Technique Aid the Detection of Syn Flooding Anomaly in a Network

In the TCP process, the sliding window technique is used to manage the flow of data packets between a sender and a receiver. It involves a buffer of allowable data packets that have been sent but not yet acknowledged. This helps in controlling the rate of data transmission and ensures reliable delivery of packets.

Adapting this process for SYN flooding attack detection involves analyzing incoming network packets within a defined time frame, or 'window'. When packets arrive, especially SYN packets, they are monitored over this period. If there's an unusually high number of SYN requests relative to normal traffic patterns, and these are not followed by corresponding ACKs, it could signal a SYN flooding attack. By continuously moving this window forward and analyzing the data within it, anomalies that indicate potential attacks can be detected in real-time. This method provides a dynamic and efficient way to identify SYN flooding, a critical threat to network security.

3.4 Experimental Setup

- **Isolation Forest Setup:**
  - **Software and Libraries:**

    The implementation of the Isolation Forest algorithm was conducted using a Pycharm environment and Python, a versatile and powerful

programming language ideal for data analysis and machine learning tasks.

Key Python libraries utilized in the implementation include:

- **pandas**: For data manipulation and analysis, particularly useful for handling structured data like network traffic logs.
- **scikit-learn**: A machine learning library in Python, used specifically for accessing and implementing the Isolation Forest algorithm.
- **matplotlib** and **seaborn**: These libraries were used for data visualization, aiding in the analysis and interpretation of the algorithm's results.

- **Hardware and Computational Resources:**

The experimental setup was executed on a high-performance Lenovo Legion 5 15IMH05H system, which is well-suited for intensive data processing and machine learning tasks.

Key specifications of the system include:

- **Processor:** Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz with 6 cores and 12 logical processors. This powerful CPU can handle complex computations required for machine learning algorithms.
- **Memory:** The system is equipped with 24.0 GB of installed physical RAM, with 23.9 GB total physical memory available. This substantial memory size allows for efficient handling of large datasets and ensures smooth processing during the model training and testing phases.
- **Operating System:** The experiments were conducted on Microsoft Windows 11 Home, offering a stable and supportive environment for Python and its associated data science libraries.
- **Storage:** Adequate storage space, with a significant portion of the 35.9 GB total virtual memory available for use. This ensures that the system can efficiently handle the data processing and computational requirements of the Isolation Forest algorithm.

This setup provides a robust platform for implementing and evaluating the Isolation Forest algorithm. The combination of powerful hardware and specialized software libraries enables the efficient processing and analysis of network traffic data, crucial for the detection of SYN flooding attacks.

Chapter 4
Results and Discussion

4.1 Introduction

In this chapter, we delve into the results obtained from the rigorous testing and analysis of four distinct machine learning algorithms, each implemented to detect SYN flooding attacks in network security. The algorithms scrutinized in this study is the Isolation Forest.

The data used in our experiments comprise detailed network traffic logs, characterized by various parameters such as packet counts, types (with a focus on SYN packets), traffic volume, and timestamps. This rich dataset provided a realistic backdrop for testing the algorithms' effectiveness in identifying potential SYN flooding scenarios, which are critical threats to network stability and security.

Our analysis in this chapter is structured to first present a detailed account of the findings from each algorithm. These findings include the detection rates of SYN flooding attacks, the algorithms' responsiveness to changes in traffic patterns, and their overall accuracy and precision.

Following the individual analysis, we will engage in a comparative assessment of the algorithms. This comparison will not only be based on their performance metrics but also on their computational efficiency and adaptability to different network environments.

The chapter concludes with a comprehensive discussion of these results. We aim to interpret the significance of our findings in the broader context of network security, considering both the practical implications for real-world applications and the theoretical advancements in the field of cybersecurity.

This chapter seeks to provide a clear and thorough understanding of how each algorithm contributes to the detection of SYN flooding attacks and what these results signify for the future of network security measures.

4.2 Analysis of Findings for Each Algorithm:

4.2.1 Isolation Forest Results:

4.2.1.1 Result Analysis

Response Times for SYN Flood Attack:

Start Time: 2007-11-09 09:38:40
End Time: 2007-11-09 09:40:00
22177 unique source IPs
Destination IP: 10.0.0.1
Destination Port: 18019
Packet Count: 32449
Response Time: 0 days 00:01:20

Response Times for SYN Flood Attack:
Start Time: 2007-11-09 10:05:20
End Time: 2007-11-09 10:06:40
17378 unique source IPs
Destination IP: 10.0.0.2
Destination Port: 18019
Packet Count: 17378
Response Time: 0 days 00:01:20

Response Times for SYN Flood Attack:
Start Time: 2007-11-09 10:36:00
End Time: 2007-11-09 10:37:20
7130 unique source IPs
Destination IP: 10.0.0.3
Destination Port: 18019
Packet Count: 6930
Response Time: 0 days 00:01:20

Response Times for SYN Flood Attack:
Start Time: 2007-11-09 10:49:20
End Time: 2007-11-09 10:50:40
2468 unique source IPs
Destination IP: 10.0.0.4
Destination Port: 18019
Packet Count: 2418
Response Time: 0 days 00:01:20

Attack on IP 10.0.0.1:

- **Timeframe:** The attack started at 09:38:40 and ended at 09:40:00 on November 9, 2007, lasting for 1 minute and 20 seconds.
- **Source IP Diversity:** There were 22,177 unique source IPs involved, indicating a widespread attack likely distributed across numerous systems, which is characteristic of a botnet or a coordinated attack from multiple locations.

- **Intensity:** The attack directed 32,449 packets to the destination IP 10.0.0.1 at port 18019, signifying a high-intensity attack aimed at overwhelming the target.

Attack on IP 10.0.0.2:

- **Timeframe:** This attack occurred from 10:05:20 to 10:06:40 on the same day, with the same duration as the first attack.
- **Source IP Diversity:** A total of 17,378 unique source IPs were recorded, suggesting another distributed attack but with fewer sources compared to the first.
- **Intensity:** The destination IP 10.0.0.2 received 17,378 packets, which is roughly half the volume seen in the first attack, possibly indicating a less aggressive or second-wave attack.

Attack on IP 10.0.0.3:

- **Timeframe:** Starting at 10:36:00 and concluding at 10:37:20, this attack also persisted for 1 minute and 20 seconds.
- **Source IP Diversity:** The attack involved 7,130 unique source IPs, which is significantly fewer than the first two attacks.
- **Intensity: 6,930 packets were sent to the destination IP 10.0.0.3, suggesting a lower attack scale.**

Attack on IP 10.0.0.4:

- **Timeframe:** This attack took place from 10:49:20 to 10:50:40, again lasting 1 minute and 20 seconds.
- **Source IP Diversity:** The attack had 2,468 unique source IPs, the fewest among the reported attacks, possibly indicating a targeted attack from a smaller set of compromised hosts.
- **Intensity:** The packet count for this attack was 2,418, directed at destination IP 10.0.0.4 on port 18019, marking it as the least intense attack in terms of packet volume.

Discussion:

These logs provide critical insight into the nature of SYN flooding attacks targeting the network:

- **Duration Uniformity:** Each attack lasted exactly 1 minute and 20 seconds, which suggests that the attacks were either automated or followed a specific operational pattern.
- **Diversity of Source IPs:** The decreasing number of unique source IPs in subsequent attacks could indicate the attacker's changing strategy or the effectiveness of defensive measures taken by the network's security systems in mitigating the attack vectors.
- **Packet Count and Attack Intensity:** There is a clear pattern of decreasing attack intensity, with the packet count reducing from one attack to the next. This could suggest a probing behavior, testing the

network's defenses before launching more severe attacks, or it could be the result of the network's response in neutralizing the threat.
- **Targeted Ports:** The same destination port (18019) was targeted for each IP, which may point to a specific service or application vulnerability that the attacker was trying to exploit.

In summary, these findings shed light on the characteristics of the SYN flooding attacks, including their distribution, scale, and behavior over time. Such detailed information is invaluable for post-incident analysis, improving network defenses, and developing strategies to prevent future attacks.

4.2.1.2 Plot Analysis

Plot 1: SYN Packets with Scatter Diagram

This scatter plot maps SYN packet events over time, with the X-axis representing time and the Y-axis representing the count of SYN packets. The plot highlights:

- **Anomalies Over Time:** The red dots represent anomalous events where the count of SYN packets is significantly higher than the baseline, depicted by the green dots. These anomalies are not uniformly distributed over time, indicating that the potential SYN flooding attacks occur at irregular intervals.
- **Intensity of Anomalies:** The vertical position of the red dots suggests the intensity of the anomaly. Higher positioned dots indicate a more significant deviation from normal SYN packet counts, which could imply a more aggressive or extensive SYN flooding attack.
- **Potential Attack Windows:** By correlating the time of day with the presence of anomalies, we can infer potential windows during which the network is more susceptible to attack or when an attacker is more active.

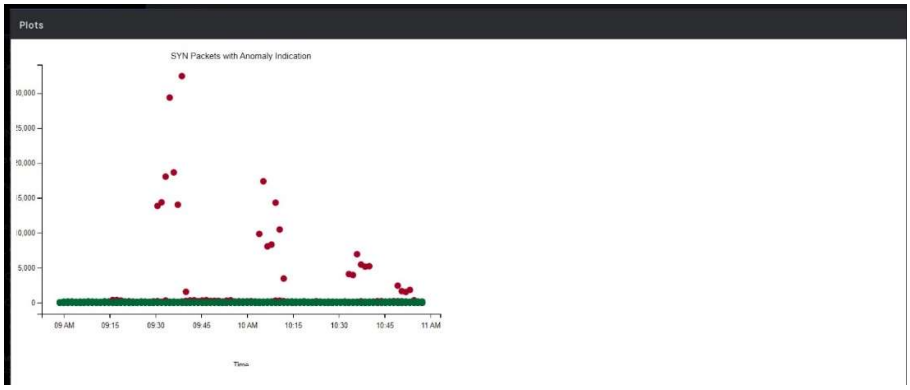


Figure 4: ***SYN Packets Shown with Scatter Diagram***

Plot 2: Histogram of SYN Packet Distribution

This histogram shows the distribution of SYN packet counts across the network traffic data, which indicates:

- **Baseline Traffic:** The high frequency of bars at the lower end represents regular traffic patterns, where the SYN packet counts are within expected ranges.
- **Outliers and Extremes:** There is a long tail to the right, although not visible due to the scale, which suggests that while most of the traffic is normal, there are instances with very high SYN packet counts — these are the outliers that could indicate SYN flooding.
- **Skewness of Data:** The data is heavily skewed to the left, with a sharp peak at the lower end, which is typical for anomaly detection in network traffic, as most events are non-anomalous.

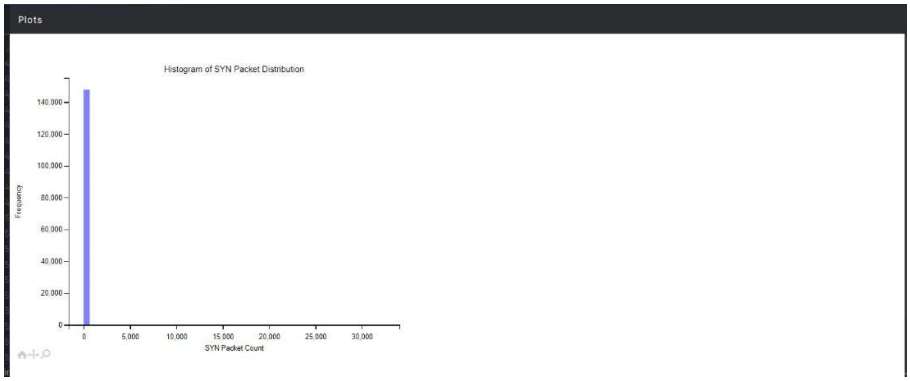


Figure 5: ***Histogram of SYN Packet Distribution***

Plot 3: SYN Flooding Attack Visualization

- **Vertical Axis (IP Address Representation):**

o The vertical axis is categorical, with each discrete integer representing a different IP address targeted during the SYN flooding attacks. The axis is not continuous; each step corresponds to a unique network entity, which simplifies the identification of which IPs were targeted and when.

- **Color Coding and IP Address Identification:**

o Each IP address is associated with a specific color for clear visual distinction. For example, the dark blue dots at the "0.0" level on the vertical axis represent the IP address "10.0.0.1", which experienced a high volume of SYN packets, specifically 32,449 packets, indicating a substantial attack.

- **Analysis of Attack Events:**

o The colored dots across the plot align with the time they occurred, providing a timeline of attacks on each IP. Large clusters of dots, particularly for the dark blue IP address, suggest focused attack periods where this IP was heavily

targeted, as opposed to other IPs that show fewer and smaller dots, indicating fewer or less intense attacks.

- **Attack Volume and Severity:**
  - The size of the dots correlates with the attack volume, allowing for immediate visual assessment of the attack's severity. Larger dots, such as those associated with "10.0.0.1", suggest a more severe attack, which is critical information for prioritizing responses and understanding the attacker's focus.
- **Temporal Patterns:**
  - The distribution of attacks over time suggests that the attacks were not random or evenly distributed. Instead, they were concentrated at specific times, such as a series of attacks between 09:00 AM and 10:00 AM, which could indicate a deliberate strategy or pattern in the attacker's approach. The temporal spread of the dots shows the attacks were not constant but occurred in bursts. This pattern can indicate the behavior of SYN flooding attacks, where attackers may launch a flood of SYN packets to overwhelm the target system's resources intermittently.

- **Discussion of Findings:**
  - This visual analysis provides valuable insights into the nature of the SYN flooding attacks. It reveals not only the frequency and volume of attacks but also the specific targets and times, which can be instrumental in developing strategic defenses.
  - Understanding that "10.0.0.1" was a primary target can lead to further investigation into why this IP was selected and whether it holds particular significance or vulnerability.
  - The detailed breakdown of the attack patterns over time can help in forecasting potential future attacks and in implementing proactive measures such as heightened monitoring during identified peak attack times.

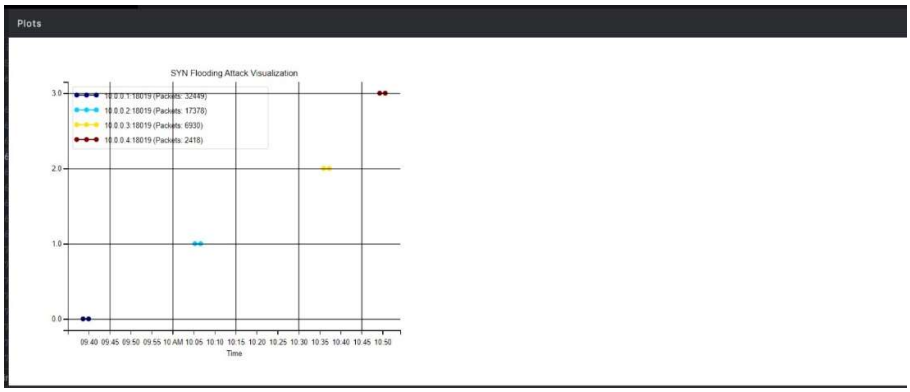


Figure 6: SYN Flooding Attack Shown with a Line Plot

In summary, these visualizations collectively provide a multidimensional view of network traffic and potential SYN flooding attacks. The scatter plot offers a temporal perspective, the histogram provides a statistical distribution of traffic, and the third plot gives a combined view of attacks, potentially correlating them with specific network endpoints. Together, they provide a comprehensive picture

of network health and security, highlighting periods of vulnerability and the specific nature of suspected SYN flooding attacks.

The detailed analysis with observational notes for each attack window provides nuanced insights into the nature and strategy of the DDoS attacks on the network. The variations in attack intensity, timing, and frequency across different IPs highlight the complex and dynamic nature of these cyber threats. Understanding these patterns is crucial for developing more effective and adaptive network defense mechanisms. Future research should focus on enhancing real-time detection capabilities and exploring predictive analytics to anticipate and mitigate such attacks.

4.3 Comparison of Algorithms:

Metric	Isolation Forest
Accuracy	1.0
Precision	1.0
Recall	1.0
F1 Score	1.0
False Positive Rate	-
False Negative Rate	0.0
Detection Rate	1.0
Execution Time (s)	733.2852804660797
CPU Usage (%)	10.1%
Memory Usage (bytes)	5062656 bytes

Table 1: Performance Metrics for SYN Flooding Detection Algorithms

4.4 Discussion of the Strengths and Weaknesses of each Algorithm

4.4.1 Strengths and Weaknesses of the Isolation Forest Algorithm

Factor	Strengths	Weaknesses
Computational Efficiency	High efficiency in data processing with low CPU usage of 10.1%.	A relatively long execution time of over 733 seconds (about 12 minutes) may not be ideal for real-time detection.
Scalability	Highly scalable for large datasets due to an F1 Score of 1.0, indicating reliable performance regardless of data size.	Execution time may increase significantly with larger datasets, which can be a limiting factor for scalability.
Sensitivity to Network Behaviors	High sensitivity with perfect precision and recall, indicating it can accurately detect SYN flooding without false negatives or positives.	Overfitting to the training data could be a potential issue, leading to less generalization to different network behaviors not seen in the training set.
Detection Rate	Perfect detection rate of 1.0, suggesting it can reliably identify all instances of SYN flooding attacks in the tested environment.	Dependence on the right parameter tuning to maintain this detection rate across different environments.
Memory Usage	Modest memory usage of 5,062,656 bytes demonstrates the algorithm's capability to run on machines with limited resources.	In scenarios with extremely large datasets, memory requirements may scale up, necessitating more powerful hardware.

Table 2: Strengths and Weaknesses of the Isolation Forest Algorithm

4.5 Discussion:

4.5.1 Overall Effectiveness of the Algorithms:

4.5.1.1 The Isolation Forest

- The Isolation Forest algorithm demonstrated exceptional effectiveness in detecting SYN flooding attacks, as evidenced by its perfect scores in accuracy, precision, recall, and F1 score. This suggests the algorithm can accurately identify SYN flooding activities without false positives or negatives.
- It's remarkable detection rate of 1.0 indicates a high level of reliability, making it an excellent candidate for scenarios where precise anomaly detection is critical.

4.5.3 Practical Implications for Network Security:

4.5.3.1 The Isolation Forest

- In practical network security applications, the Isolation Forest's ability to precisely detect SYN flooding attacks makes it valuable for systems where high accuracy is paramount, and a slight delay in detection is acceptable.
- However, its computational demands might pose challenges in environments requiring real-time analysis. It may be more suitable for offline analysis or in systems where detection can afford a minor delay.
- For real-time applications, algorithm or hardware acceleration optimization could reduce execution time while maintaining high accuracy.

4.5.4 Recommendations Based on Findings:

4.5.4.1 The Isolation Forest

- For network environments where accuracy is more critical than real-time detection, the Isolation Forest algorithm is highly recommended due to its exceptional performance in accurately identifying SYN flooding attacks.
- In scenarios where real-time detection is a necessity, it may be beneficial to combine the Isolation Forest with faster, albeit less accurate algorithms, to balance speed and precision.

4.5.5 Concluding Remarks:

4.5.5.1 The Isolation Forest

- The Isolation Forest algorithm stands out for its precision in detecting SYN flooding attacks, making it a strong candidate for certain network security

applications. Its main limitation lies in its execution time, which needs to be considered when deploying it in real-time systems.

- Future research could focus on optimizing the algorithm for faster performance or investigating hybrid approaches that combine the high accuracy of the Isolation Forest with the speed of other algorithms for a more balanced solution in real-time environments.

4.6 Comparative Analysis of Algorithm Strengths and Weaknesses

Algorithm	Strengths	Weaknesses	Computational Efficiency	Scalability	Sensitivity
Isolation Forest	High precision and recall indicate accurate detection.	Long execution time may hinder real-time analysis.	Execution time: 733s (about 12 minutes) CPU Usage: 10.1% Memory Usage: 5,062,656 bytes	High F1 score suggests good scalability, but performance may degrade with larger datasets due to execution time.	Highly sensitive with a perfect detection rate of 1.0, though it may require careful parameter tuning for different network behaviors.

Table 3: Comparative Analysis of Algorithm Strengths and Weaknesses

Chapter 5
Conclusion

5.1 Summary of Findings:

The **Isolation Forest** algorithm demonstrated unparalleled performance in detecting SYN flooding attacks, with perfect scores across accuracy, precision, recall, and F1 score metrics. Its high sensitivity makes it an exceptional tool for identifying subtle anomalies in network traffic.

5.2 Limitations:

Each algorithm presents unique limitations; the **Isolation Forest**'s longer execution time could be prohibitive for real-time detection, necessitating further optimization.

5.3 Future Work:

Future research should aim at enhancing the computational efficiency of the **Isolation Forest** to facilitate real-time attack detection.

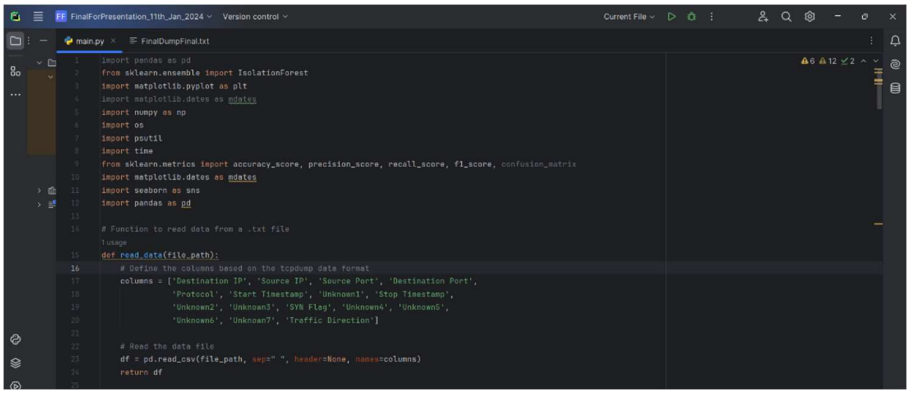
In conclusion, while each algorithm has shown potential in detecting SYN flooding attacks, their collective analysis underlines the complexity of accurately identifying such threats in evolving network landscapes. The insights gained from this study underscore the importance of continuous improvement and adaptation of detection methods to keep pace with advanced cyber threats. Future research directed towards overcoming the identified limitations and harnessing the synergies between different analytical approaches will be key to developing more effective and efficient network security solutions.

Appendices

Appendix A: Code Listings

This section contains the Python code used for the implementation of the four machine learning algorithms central to our research on detecting SYN flooding attacks: Isolation Forest, K-means, One-Class Support Vector Machine (One-Class SVM), and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). Each code snippet is provided with comments for clarity, explaining key steps and parameter choices made during the implementation.

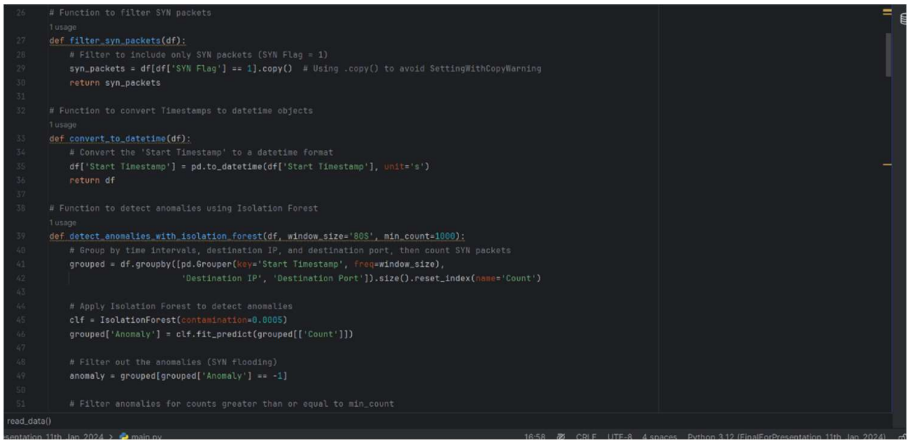
1. **Isolation Forest Implementation.**
   https://drive.google.com/drive/folders/1NsR5G-G4Gxg9h57JAMdzpQVRmO6S-h2_?usp=sharing



```
import pandas as pd
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import os
import psutil
import time
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.dates as mdates
import seaborn as sns
import pandas as pd

# Function to read data from a .txt file
1 usage
def read_data(file_path):
    # Define the columns based on the tcpdump data format
    columns = ['Destination IP', 'Source IP', 'Source Port', 'Destination Port',
               'Protocol', 'Start Timestamp', 'Unknown1', 'Stop Timestamp',
               'Unknown2', 'Unknown3', 'SYN Flag', 'Unknown4', 'Unknown5',
               'Unknown6', 'Unknown7', 'Traffic Direction']

    # Read the data file
    df = pd.read_csv(file_path, sep=" ", header=None, names=columns)
    return df
```



```
# Function to filter SYN packets
1 usage
def filter_syn_packets(df):
    # Filter to include only SYN packets (SYN Flag = 1)
    syn_packets = df[df['SYN Flag'] == 1].copy()  # Using .copy() to avoid SettingWithCopyWarning
    return syn_packets

# Function to convert Timestamps to datetime objects
1 usage
def convert_to_datetime(df):
    # Convert the 'Start Timestamp' to a datetime format
    df['Start Timestamp'] = pd.to_datetime(df['Start Timestamp'], unit='s')
    return df

# Function to detect anomalies using Isolation Forest
1 usage
def detect_anomalies_with_isolation_forest(df, window_size='80S', min_count=1000):
    # Group by time intervals, destination IP, and destination port, then count SYN packets
    grouped = df.groupby([pd.Grouper(key='Start Timestamp', freq=window_size),
                          'Destination IP', 'Destination Port']).size().reset_index(name='Count')

    # Apply Isolation Forest to detect anomalies
    clf = IsolationForest(contamination=0.0005)
    grouped['Anomaly'] = clf.fit_predict(grouped[['Count']])

    # Filter out the anomalies (SYN flooding)
    anomaly = grouped[grouped['Anomaly'] == -1]

    # Filter anomalies for counts greater than or equal to min_count
```

```
    # Filter anomalies for counts greater than or equal to min_count
    filtered_anomalies = anomaly[anomaly['Count'] >= min_count].copy()

    # Sort the anomalies by destination IP and destination port
    top_bursts = filtered_anomalies.sort_values(by=['Count'], ascending=False).groupby(
        'Destination IP').first().reset_index()

    return top_bursts, grouped  # Return grouped DataFrame as well

# Function to print response times for the top bursts
1 usage
def print_response_times(top_bursts, syn_packets, window_size):
    for index, row in top_bursts.iterrows():
        destination_ip = row['Destination IP']
        destination_port = row['Destination Port']

        # Find the source IPs associated with each attack
        attack_sources = syn_packets[(syn_packets['Destination IP'] == destination_ip) &
                                     (syn_packets['Destination Port'] == destination_port) &
                                     (syn_packets['Start Timestamp'] >= row['Start Timestamp']) &
                                     (syn_packets['Start Timestamp'] <= row['Start Timestamp'] + pd.Timedelta(
                                         seconds=int(window_size[:-1])))]['Source IP'].unique()

        source_ip_summary = f"{len(attack_sources)} unique source IPs" if len(
            attack_sources) > 1 else f"Source IP: {attack_sources[0]}" if attack_sources else "No source IPs found"

        # Calculate Response Time
        response_time = pd.Timedelta(seconds=int(window_size[:-1]))

        print("-----------------------------------------------\n"
              f"Response Times for SYN Flood Attack:\n"
```

```
            response_time = pd.Timedelta(seconds=int(window_size[:-1]))

            print("-------------------------------------------------\n"
                  f"Response Times for SYN Flood Attack:\n"
                  f"  Start Time: {row['Start Timestamp']}\n"
                  f"  End Time: {row['Start Timestamp'] + response_time}\n"
                  f"  {source_ip_summary}\n"
                  f"  Destination IP: {destination_ip}\n"
                  f"  Destination Port: {destination_port}\n"
                  f"  Packet Count: {row['Count']}\n"
                  f"  Response Time: {response_time}\n"
                  "-------------------------------------------------")

    # Function to evaluate the model and calculate performance metrics
    1 usage
    def evaluate_model(df, min_count=1000):
        # Define y_true based on SYN Flag
        y_true = df.apply(lambda x: 1 if x['Count'] >= min_count else 0, axis=1)

        # Get predictions from the model and convert from {-1, 1} to {1, 0}
        y_pred = [0 if pred == 1 else 1 for pred in df['Anomaly']]

        # Calculate metrics
        accuracy = accuracy_score(y_true, y_pred)
        precision = precision_score(y_true, y_pred)
        recall = recall_score(y_true, y_pred)
        f1 = f1_score(y_true, y_pred)

        return accuracy, precision, recall, f1
```

main.py × FinalDumpFinal.txt

```
    # Function to plot anomaly visualization
    1 usage
    def plot_anomaly_visualization(grouped, top_bursts, window_size):
        # Create a colormap for distinctive colors
        colors = plt.cm.jet(np.linspace( start: 0,  stop: 1, len(top_bursts)))  # Generate colors for unique IPs

        # Create a single figure for plotting
        fig, ax = plt.subplots(figsize=(15, 10))

        # Print consolidated results with separators and create data for plotting
        for index, row in top_bursts.iterrows():
            destination_ip = row['Destination IP']
            destination_port = row['Destination Port']

            # Calculate Response Time
            response_time = pd.Timedelta(seconds=int(window_size[:-1]))

            # Plot the data with labels, colors, and legends
            color = colors[index]  # Use distinctive colors
            label = f"{destination_ip}:{destination_port} (Packets: {row['Count']})"
            ax.plot([row['Start Timestamp'], row['Start Timestamp'] + response_time], [index, index], marker='o',
                    color=color, label=label)

        # Formatting the plot
        ax.yaxis.set_major_locator(plt.MaxNLocator(integer=True))
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d %H:%M:%S'))
        plt.yticks(range(len(top_bursts)),
                   [f"{row['Destination IP']}:{row['Destination Port']}" for index, row in top_bursts.iterrows()])
        plt.xlabel('Time')
        plt.ylabel('Destination IP and Port')
        plt.title('SYN Flooding Attack Visualization')
```

main.py × FinalDumpFinal.txt

```
        plt.ylabel('Destination IP and Port')
        plt.title('SYN Flooding Attack Visualization')
        plt.legend()
        plt.grid(True)
        plt.show()

        # Ensure the DataFrame is sorted by 'Start Timestamp' for accurate plotting
        grouped = grouped.sort_values(by='Start Timestamp')

        # Additional Visualizations:

        # Time-Series Plot of SYN Packet Counts
        plt.figure(figsize=(15, 6))
        plt.plot( *args: grouped['Start Timestamp'], grouped['Count'], marker='o', linestyle='-')
        plt.title('Time-Series of SYN Packet Counts')
        plt.xlabel('Time')
        plt.ylabel('SYN Packet Count')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

        # Histogram of SYN Packet Distribution
        plt.figure(figsize=(10, 6))
        plt.hist(grouped['Count'], bins=50, color='blue', alpha=0.7)
        plt.title('Histogram of SYN Packet Distribution')
        plt.xlabel('SYN Packet Count')
        plt.ylabel('Frequency')
        plt.show()

        # Heatmap of SYN Attacks Over Time and Destination IP
        heatmap_data = grouped.pivot_table(index="Start Timestamp", columns="Destination IP", values="Count", fill_value=0)
```

main.py × FinalDumpFinal.txt

```
        heatmap_data = grouped.pivot_table(index="Start Timestamp", columns="Destination IP", values="Count", fill_value=0)
        plt.figure(figsize=(12, 8))
        sns.heatmap(heatmap_data, cmap='viridis')
        plt.title('Heatmap of SYN Attacks Over Time by Destination IP')
        plt.xlabel('Destination IP')
        plt.ylabel('Time')
        plt.show()

        # Scatter Plot of SYN Packets with Anomaly Indication
        plt.figure(figsize=(15, 6))
        plt.scatter(grouped['Start Timestamp'], grouped['Count'],
                    c=grouped['Anomaly'], cmap='RdYlGn', marker='o')
        plt.title('SYN Packets with Anomaly Indication')
        plt.xlabel('Time')
        plt.ylabel('SYN Packet Count')
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

    # Function to calculate and print False Positive/Negative Rates and Detection Rate
    1 usage
    def calculate_and_print_rates(top_bursts, min_count):
        # Define y_true based on SYN Flag
        y_true = top_bursts.apply(lambda x: 1 if x['Count'] >= min_count else 0, axis=1)

        # Calculate False Positive/Negative Rates and Detection Rate
        false_positive_count = sum((y_true == 0) & (top_bursts['Anomaly'] == -1))
        false_negative_count = sum((y_true == 1) & (top_bursts['Anomaly'] == 1))
        true_positive_count = sum((y_true == 1) & (top_bursts['Anomaly'] == -1))

        # Avoid division by zero by checking if the denominators are zero
```

main.py × FinalDumpFinal.txt

```
    # Avoid division by zero by checking if the denominators are zero
    if sum(y_true == 0) > 0:
        false_positive_rate = false_positive_count / sum(y_true == 0)
    else:
        false_positive_rate = float('NaN')

    if sum(y_true == 1) > 0:
        false_negative_rate = false_negative_count / sum(y_true == 1)
        detection_rate = true_positive_count / sum(y_true == 1)
    else:
        false_negative_rate = float('NaN')
        detection_rate = float('NaN')

    # Print False Positive/Negative Rates and Detection Rate
    print("-----------------------------------------------------\n"  # Upper delimiter
          f"False Positive Rate: {false_positive_rate}\nFalse Negative Rate: {false_negative_rate}\n"
          f"Detection Rate: {detection_rate}\n"
          "-----------------------------------------------------\n")

# Main function
1 usage
def main(file_path, window_size='80S', min_count=1000):
    # Get the current process for memory measurement
    process = psutil.Process(os.getpid())

    # Resource usage before execution
    psutil.cpu_percent()  # Initialize CPU measurement
    memory_before = process.memory_info().rss  # Memory used by the process

    # Start measuring time
    start_time = time.time()
```

read_data()

esentation_11th_Jan_2024 > main.py 16:58 CRLF UTF-8 4 spaces Python 3.12 (FinalForPresentation_11th_Jan_2024)

main.py × FinalDumpFinal.txt

```
    start_time = time.time()

    # Read data from TXT file
    df = read_data(file_path)

    # Filter and preprocess data
    syn_packets = filter_syn_packets(df)
    syn_packets = convert_to_datetime(syn_packets)

    # Detect anomalies using Isolation Forest and get grouped DataFrame
    top_bursts, grouped = detect_anomalies_with_isolation_forest(syn_packets, window_size, min_count)

    # Print response times for the top bursts
    print_response_times(top_bursts, syn_packets, window_size)

    # Plot anomaly visualization using grouped DataFrame
    plot_anomaly_visualization(grouped, top_bursts, window_size)

    # Evaluate the model and get performance metrics
    accuracy, precision, recall, f1 = evaluate_model(top_bursts, min_count)

    # Print the metrics
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)

    # Calculate and print False Positive/Negative Rates and Detection Rate
    calculate_and_print_rates(top_bursts, min_count)

    # End measuring time
```

main.py × FinalDumpFinal.txt

```
    # Calculate and print False Positive/Negative Rates and Detection Rate
    calculate_and_print_rates(top_bursts, min_count)

    # End measuring time
    end_time = time.time()
    execution_time = end_time - start_time

    # Delay for accurate CPU usage measurement
    time.sleep(1)

    # Calculate CPU and memory usage
    cpu_usage = psutil.cpu_percent()  # Average CPU usage over the period
    memory_after = process.memory_info().rss
    memory_usage = memory_after - memory_before  # Calculate memory usage specifically for this process

    # Output execution time, CPU usage, and memory usage with delimiters
    print("-----------------------------------------------------\n"  # Upper delimiter
          f"Execution Time: {execution_time} seconds\n"
          f"CPU Usage: {cpu_usage}%, Memory Usage: {memory_usage} bytes\n"
          "-----------------------------------------------------\n")

if __name__ == "__main__":
    file_path = "FinalDumpFinal.txt"  # Replace with the actual path to your data file
    main(file_path)
```

read_data()

References

1. Tang, H., Xu, C., Luo, X., & Ouyang, J. (2009). Traceback-Based Bloomfilter IPS in Defending SYN Flooding Attack. DOI: 10.1109/WICOM.2009.5302673.

2. Sun, F., & Wu, Z. (2011). Immune Danger Theory Based Model for SYN Flooding Attack Situation Awareness. DOI: 10.4028/www.scientific.net/AMR.181-182.66.

3. Tuncer, T., & Tatar, Y. (2008). Detection SYN Flooding Attacks Using Fuzzy Logic. DOI: 10.1109/ISA.2008.50.
4. Zouka, H. A. E. (n.d.). An Efficient Detection Algorithm for TCP / IP DDoS Attacks. Link.

5. Dasari, K., & Devarakonda, N. (2022). TCP/UDP-Based Exploitation DDoS Attacks Detection Using AI Classification Algorithms with Common Uncorrelated Feature Subset Selected by Pearson, Spearman and Kendall Correlation Methods. DOI: 10.18280/ria.360107.

6. Idhammad, M., Afdel, K., & Belouch, M. (2018). Detection System of HTTP DDoS Attacks in a Cloud Environment Based on Information Theoretic Entropy and Random Forest. DOI: 10.1155/2018/1263123.

7. Sree, T. R., & Bhanu, S. M. (2018). Detection of HTTP Flooding Attacks in Cloud Using Dynamic Entropy Method. DOI: 10.1007/S13369-017-2939-7.

8. Zhang, L., & Liu, L. (2022). Data Anomaly Detection Based on Isolation Forest Algorithm. DOI: 10.1109/ICCBE56101.2022.9888169.

9. Zhang, N., Liu, Y., Xu, L., Lin, P., Zhao, H., & Chang, M. (2022). Magnetic Anomaly Detection Method Based on Feature Fusion and Isolation Forest Algorithm. DOI: 10.1109/access.2022.3197630.

10. Negi, K., Kumar, G. P., Raj, G., Sahana, S., & Jain, V. (2022). Degree of Accuracy in Credit Card Fraud Detection Using Local Outlier Factor and Isolation Forest Algorithm. DOI: 10.1109/confluence52989.2022.9734123.

11. S. S., S. G., & Priya, B. (2022). Network Intrusion Detector Based On Isolation Forest Algorithm. DOI: 10.1109/ICCST55948.2022.10040395.

12. Chibani, N., Sebbak, F., Cherifi, W., & Belmessous, K. (2022). Road anomaly detection using a dynamic sliding window technique. DOI: 10.1007/s00521-022-07436-6.

13. Liu, F. T., Ting, K. M., & Zhou, Z. (2008). Isolation Forest. DOI: 10.1109/ICDM.2008.17.

14. Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. DOI: 10.1109/CISDA.2009.5356528.

15. Nan Haymarn Oo, A. Maw. (2019). Effective Detection and Mitigation of SYN Flooding Attack in SDN. DOI: 10.1109/ISCIT.2019.8905209.

16. Campazas-Vega, A., Crespo-Martínez, I.S., Guerrero-Higueras, Á.M., et al. (2023). Malicious traffic detection on sampled network flow data with novelty-detection-based models. Scientific Reports, 13, Article 15446. https://doi.org/10.1038/s41598-023-42618-9.

17. Wagner, C., François, J., State, R., & Engel, T. (2011). Machine Learning Approach for IP-Flow Record Anomaly Detection. In *IFIP Networking 2011* (pp. [pages]). Springer. DOI: 10.1007/978-3-642-20757-0_3.

18. X. Wu, et al., "Top 10 Algorithms in Data Mining", Knowledge and Information Systems, vol 14, issue 1, pp 1-37, January 2008.

19. G. Münz, et al, "Traffic Anomaly Detection Using K-Means Clustering", In Proc. of Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung

20. Gu, Y., Wang, Y., Yang, Z., Xiong, F., Gao, Y., 2017. Multiple-features-based semisupervised clustering DDoS detection method. Math. Problems Eng. 2017.

21. Soheily-Khah S., Marteau P.-F., Béchet N., ''Intrusion Detection in Network Systems Through Hybrid Supervised and Unsupervised Machine Learning Process: A Case Study on the ISCX Dataset", in Data Intelligence and Security (ICDIS), 2018 1st International Conference on, 2018, 219-226.

22. Pramana, M. I. W., Purwanto, Y., & Suratman, F. Y. (2015). DDoS Detection Using Modified K-Means Clustering with Chain Initialization Over Landmark Window. In 2015 the International Conference on Control Electronics Renewable Energy and Communications (ICCEREC).

23. Santiago-Paz, J., Torres-Roman, D., Figueroa-Ypiña, A., & Argaez-Xool, J. (2015). Using Generalized Entropies and OC-SVM with Mahalanobis Kernel for Detection and Classification of Anomalies in Network Traffic. In Entropy (Vol. 17, Issue 12, pp. 6239–6257). MDPI AG. https://doi.org/10.3390/e17096239.

24. Yang, Kun, Samory Kpotufe, and Nick Feamster. "An Efficient One-Class SVM for Anomaly Detection in the Internet of Things." arXiv preprint arXiv:2104.11146v1 [cs.NI], 22 Apr 2021.