Low and Slow Attack Detection on HTTP/2

Naor Ladani, Itamar Cohen, Tuvia Smadar
Department of Computer Science
Ariel Cyber Innovation Center
Ariel University, Israel

Abstract

Detecting low and slow attacks in HTTP/2 traffic is a challenging task due to their stealthy nature, where malicious packets are sent at a low rate to evade traditional detection methods. Existing detection techniques often rely on analyzing payloads or require specific traffic signatures, making them less effective for such subtle attacks. In this work, we present an approach for identifying low and slow attacks by analyzing packet captures and focusing on event sequences, packet timing, and delays to learn behaviors, and then run our checks to detect attack. Our method evaluates the inter packet intervals and timing behavior to identify deviations from normal HTTP/2 traffic patterns, which are indicative of potential attacks. Unlike existing approaches, our technique does not require prior knowledge of the payload or network-specific signatures, making it broadly applicable. Experimental evaluation demonstrates that analyzing timing and delay in HTTP/2 traffic provides an effective and lightweight method for detecting low and slow attacks, contributing to enhanced security for modern web protocols.

Keywords: HTTP/2 • Anomaly detection • Event sequence analysis • Low and slow

1 Introduction

The evolution of web technologies has led to the widespread adoption of the HTTP/2 protocol, designed to improve the performance of modern web services. HTTP/2 offers significant advancements over its predecessor (HTTP/1.1), including features such as multiplexing, header compression, and prioritized streams [1]. These enhancements enable faster data transfers, reduced latency, and efficient utilization of network resources, which are critical for the modern internet ecosystem.

However, the same features that make HTTP/2 highly efficient have also expanded its attack surface, enabling adversaries to exploit protocol behaviors in different ways [2, 3]. One such emerging threat is the class of low and slow denial-of-service (DoS) attacks, which have become a growing concern in network security [4]. These attacks involve sending small amounts of malicious traffic at extremely low rates, gradually exhausting server resources while avoiding detection by conventional network monitoring systems. Unlike traditional DoS attacks, which rely on high traffic volumes to overwhelm servers, low and slow attacks operate covertly, often masquerading as legitimate traffic.

Denial-of-Service (DoS) attack is one of the most common forms of cyberattacks, aimed at disrupting access to a service by exhausting its resources. These attacks can be categorized into two main types: volumetric attacks and low-rate attacks. Volumetric DoS attacks, including Distributed Denial-of-Service (DDoS) attacks, rely on generating a high volume of traffic to overwhelm a target. DDoS attacks amplify this approach by coordinating traffic from multiple compromised machines (botnets), making them particularly destructive [5].

In contrast, low-rate DoS attacks do not depend on traffic intensity. Instead, they exploit vulnerabilities in application-layer protocols to consume server resources subtly. For example, attackers can maintain long-lived connections or send incomplete requests, tying up server threads, memory, and other resources. This stealthy approach allows such attacks to remain undetected by rate-based detection systems, posing significant challenges for defenders [6].

Low and Slow Attacks in HTTP/2 The introduction of HTTP/2 has addressed many inefficiencies of HTTP/1.1 but has also opened new opportunities for low and slow attacks. Features such as multiplexing and flow control, which optimize legitimate traffic, can also be abused by adversaries to

maintain long-lived malicious connections. For instance, HTTP/2 allows multiple streams to share a single connection, making it difficult to distinguish benign traffic from malicious activity. Attackers can exploit these features to launch low-rate attacks that mimic legitimate behavior while gradually consuming server resources [7].

The covert nature of these attacks and the lack of effective detection mechanisms for HTTP/2 traffic make low and slow attacks particularly concerning. Traditional detection methods, such as intrusion detection systems (IDS) and payload-based inspection, often fail against these threats because they operate under normal traffic thresholds and frequently leverage encrypted communication channels.

2 Background

The evolution of web communication protocols has been crucial in improving the performance and efficiency of internet services. The introduction of HTTP/2, defined in RFC 7540 (2015), brought significant advancements over its predecessor HTTP/1.1, which had been in use for nearly two decades. HTTP/2 was developed to address several limitations of HTTP/1.1, particularly around network inefficiency and latency. One of the key improvements of HTTP/2 is its ability to handle multiple simultaneous requests over a single TCP connection using multiplexing. This innovation eliminates the head-of-line blocking problem in HTTP/1.1, where the arrival of a slow request would block the processing of subsequent requests.

Despite its advantages, HTTP/2 introduces new opportunities for attack vectors that did not exist in HTTP/1.1. Among these, Low and Slow attacks have become a growing concern. These attacks target the core features of HTTP/2, exploiting the way it handles connection management and flow control. A Low and Slow attack is designed to consume server resources at a very slow rate, often by making requests that are designed to take up server time and bandwidth while using minimal network resources. These attacks can remain undetected for extended periods due to the slow nature of their execution, making them particularly difficult to mitigate.

With HTTP/2's multiplexed streams and flow control mechanisms, attackers can open many simultaneous streams with small, slow data packets, effectively tying up the server's resources and reducing its capacity to handle legitimate traffic. Unlike traditional DoS (Denial of Service) attacks, which may flood a server with massive amounts of traffic, Low and Slow attacks can be more stealthy and persistent. They do not require a large amount of bandwidth, making them harder to detect and defend against.

A key feature of HTTP/2 that is often targeted by such attacks is the stream priority and flow control mechanisms. In an HTTP/2 connection, clients can initiate multiple streams, and they can also set priorities for those streams, affecting how server resources are allocated. By creating many low-priority streams and sending tiny chunks of data, attackers can manipulate the server's flow control mechanism to ensure that each stream is maintained for an extended period, thus consuming server resources inefficiently.

In recent years, various studies and security reports have highlighted an increase in Low and Slow attacks targeting servers running HTTP/2. Although HTTP/2 offers several benefits over HTTP/1.1, it has also revealed new challenges in network security. The ability of attackers to exploit its sophisticated connection handling mechanisms has led to an increased need for effective defense mechanisms. Various countermeasures, such as rate limiting, request throttling, and connection timeout strategies, have been proposed, but these are often not foolproof in the context of HTTP/2's design.

Understanding the nature of Low and Slow attacks on HTTP/2 is crucial for developing defense strategies to protect web servers. This paper investigates these attacks in depth, analyzes their effectiveness on HTTP/2 connections, and explores potential mitigation strategies to safeguard against such attacks in the modern web infrastructure.

3 Related Work

Low-rate or low and slow Denial-of-Service (DoS) attacks have emerged as a significant challenge in network security. Unlike traditional volumetric attacks that overwhelm servers with high traffic, low-rate DoS attacks use minimal traffic to gradually exhaust server resources, making them difficult to detect and mitigate. These attacks typically exploit protocol-level vulnerabilities to maintain long-lived

connections or send incomplete requests. Research has shown that these attacks often bypass traditional intrusion detection systems (IDS) designed for high-volume traffic by operating under normal thresholds, making them stealthy and harder to identify [8]. "Exploiting HTTP/2 Protocol Features for Low-Rate Denial-of-Service Attacks" by R. Malhotra (2017)

While traditional Distributed Denial-of-Service (DDoS) attacks leverage large-scale botnets to generate massive traffic, low-rate DoS attacks operate covertly and are less reliant on high traffic volumes. This has led to a growing interest in specialized detection techniques that target subtle attack behaviors rather than relying on traffic volume. Studies have shown that low-rate attacks can evade detection, especially in the presence of encrypted traffic, where conventional signature-based methods struggle [8].

The adoption of HTTP/2, a protocol designed to improve web performance through features such as multiplexing, header compression, and flow control, has introduced new vulnerabilities. These features, while optimizing performance, can also be exploited for malicious purposes. Researchers have investigated how attackers can use HTTP/2's multiplexing capabilities to maintain multiple streams within a single connection, simulating legitimate traffic while consuming server resources. Furthermore, the flow control and stream prioritization mechanisms can be abused to introduce delays in packet delivery, making it more challenging to detect low-rate attacks [9]. "Security Risks and Attack Strategies in HTTP/2: A Study of Low-Rate DoS Attacks" by Z. Zhao (2019)

Recent studies have highlighted the exploitation of HTTP/2 in low and slow attacks. Tripathi's work demonstrates the ability of attackers to exploit HTTP/2's multiplexing and stream prioritization features for maintaining malicious connections while mimicking legitimate traffic patterns [13]. "Slow Rate Denial of Service Attacks Against HTTP/2 and Detection" by N. Tripathi. He also proposed a real-time detection mechanism based on event sequence analysis, which monitors delays and inter-event timing to identify deviations from normal traffic patterns [14]. "Delays have Dangerous Ends: Slow HTTP/2 DoS attacks into the Wild and their Real-Time Detection using Event Sequence Analysis". [15] "PRETT2: Discovering HTTP/2 DoS Vulnerabilities via Protocol Reverse Engineering" by C. Lee's research, using protocol reverse engineering, uncovered specific vulnerabilities in HTTP/2 implementations and introduced the PRETT2 framework for discovering these issues.

In response to these challenges, several detection systems have been proposed, focusing on anomalies in connection behaviors, such as unusually long connections or delayed stream data. These systems often combine signature-based and statistical methods, but they face limitations when dealing with encrypted traffic or rapidly evolving attack strategies. As such, researchers have turned to anomaly detection and behavioral analysis to identify low and slow attacks. These methods monitor packet timings, sequencing, and inter-arrival times to detect deviations from normal traffic patterns, though they often require large datasets for training and are computationally expensive [10], "Detection of Low-Rate Denial of Service Attacks in HTTP/2 Traffic" by S. Zhang et al. and Anomaly-Based Detection of Low-Rate DoS Attacks in HTTP Traffic" by F. Yang et al.[11].

Additionally, statistical methods have been proposed for detecting low-rate DoS attacks based on packet timings and inter-arrival times. These methods aim to be lightweight and real-time, making them suitable for deployment in operational environments. However, they still face challenges in distinguishing low-rate attacks from normal traffic variations, especially in the case of HTTP/2, where multiplexing and other optimizations can mask attack behavior. "Real-Time Detection of Low-Rate Denial of Service Attacks Using Statistical Analysis" by J. Huang et al. [12]. Furthermore, new studies emphasize the need for integrating real-time anomaly detection mechanisms with HTTP/2-specific features to address the unique challenges presented by this protocol [13][14].

4 Solution

4.1 Overview of the Attack

The low-and-slow DoS attack targets the HTTP/2 protocol layer, exploiting how web servers handle incomplete requests to consume resources while remaining inconspicuous. The attack begins with an adversary sending carefully crafted, incomplete HTTP/2 requests that force the server to keep connections open for extended periods. This behavior stems from servers accommodating clients on unreliable networks, such as low-bandwidth connections.

The attack consists of two main phases: network exploitation and resource exhaustion. In the

network exploitation phase, the attacker leverages the HTTP/2 protocol's design to send seemingly harmless, incomplete requests. During the resource exhaustion phase, these requests overwhelm the server's connection queue, preventing it from processing legitimate traffic and ultimately causing a Denial of Service (DoS).

To evaluate this attack, a controlled environment was created, consisting of an attacker, a client and a target web server. This setup allowed us to simulate the attack, monitor server responses, and analyze vulnerabilities related to connection handling under various attack scenarios.

4.2 Investigating Trough Attacks Versions

To investigate the vulnerabilities of HTTP/2 servers against Slow Rate DoS attacks, we embarked on a comprehensive research initiative that involved systematically examining the potential weaknesses exploited by these attacks. Recognizing the complex interplay between protocol design and implementation, our methodology emphasized creating a highly controlled and reproducible testbed environment. This setup mimicked real-world server configurations, including various HTTP/2-compatible servers, network conditions, and client behaviors. By simulating a server, we were able to explore the dynamics of server-client interactions and identify critical stress points within the protocol's operation.

Our experimental framework began with an in-depth analysis of the HTTP/2 protocol specifications, isolating features that could potentially be manipulated to degrade server performance. Key areas of focus included the protocol's multiplexing capabilities, flow control mechanisms, and its reliance on connection management strategies. These aspects were selected based on their centrality to HTTP/2's design and their potential for misuse by attackers. By dissecting the theoretical underpinnings of the protocol, we found a set of attack scenarios designed to exploit it.

A core component of our approach was the deployment of a HTTP/2 server implementation. This heterogeneity ensured that our findings were not biased toward a particular implementation. The server was configured with realistic settings. This realism was crucial for assessing the practical implications of each type of attack.

The testbed environment incorporated traffic generators capable of simulating both benign and malicious behaviors. Benign traffic patterns were modeled to reflect typical usage scenarios. In contrast, malicious traffic was crafted to embody the characteristics of Slow Rate DoS attacks. This dual-pronged traffic allowed us to observe how servers responded to legitimate requests under the strain of concurrent attack traffic.

One of the key innovations in our research was the application of event sequence analysis. By treating server-client interactions as sequences of events, we were able to detect deviations from normal patterns indicative of an ongoing attack. This approach involved constructing a database of characteristic normal event sequences during a learning phase and comparing incoming sequences against this database during the detection phase. Anomalies were flagged when significant mismatches were identified, allowing for real-time attack detection.

Our experiments revealed several critical insights. First, the susceptibility of servers to Slow Rate DoS attacks varied significantly depending on their configuration and the specific attack type. Servers with overly permissive timeout settings or inefficient resource allocation strategies were particularly vulnerable. Second, the stealthy nature of these attacks posed challenges for traditional detection mechanisms, as they generated minimal network traffic and mimicked benign behaviors.

These findings underscore the importance of adopting a good approach to server security that combines configuration practices, real-time anomaly detection, and adaptive resource management. By systematically investigating the vulnerabilities of HTTP/2 servers, our research provides a foundation for developing more resilient systems capable of withstanding the evolving threat landscape posed by Slow Rate DoS attacks.

4.2.1 Attack Version 1

The Advertising Zero Window Size attack exploits the TCP flow control mechanism by setting the advertised window size to zero. This parameter, part of the TCP header, is used by the client to inform the server of the amount of data it can receive without being overwhelmed. By advertising a zero window size, the attacker signals that it cannot receive any more data. Consequently, the server halts data transmission and keeps the connection open, anticipating that the client will eventually update the window size to a positive value.

This attack leverages the robustness principle of TCP, which prioritizes uninterrupted communication and assumes clients act in good faith. While waiting for the window size to change, the server allocates resources such as memory and processing power to maintain the connection state. If multiple such connections are initiated simultaneously, the cumulative resource consumption can degrade server performance or lead to a denial of service (DoS).

From a technical perspective, the attacker achieves this by sending a series of SYN packets to establish TCP connections and following them up with packets advertising the zero window size. These connections are kept alive through periodic zero-window probes or acknowledgments sent by the attacker. This behavior ensures that the server remains engaged with these malicious connections. The attack requires minimal computational effort from the attacker as it exploits the server's willingness to wait indefinitely for the window size to update.

Mitigation strategies include implementing idle connection timeouts and using heuristics to identify abnormal connection behaviors. For example, servers can monitor the duration and frequency of zero-window states and terminate connections exhibiting suspicious patterns. Additionally, rate-limiting new connections and employing resource management algorithms can help prevent server resource exhaustion.

4.2.2 Attack Version 2

The Incomplete POST Request Message Body attack targets the server's handling of POST requests, which typically involve transmitting large amounts of data in the message body. In this attack, the attacker sends the initial headers of a POST request but deliberately withholds the message body. HTTP/2 servers, following the protocol's specifications, wait for the complete message body to arrive before processing the request. This waiting period consumes server resources, including memory and connection slots.

The attack begins by initiating a POST request to the target server. The attacker crafts the headers to appear legitimate, specifying attributes such as Content-Length to indicate the expected size of the message body. Once the server acknowledges the headers and allocates resources to receive the body, the attacker halts further communication, leaving the connection in a pending state. The server continues to wait, often for a timeout period specified in its configuration.

This attack is particularly damaging when executed at scale, as the server's capacity to handle incoming requests becomes saturated. Legitimate clients are denied service as the server's resources are tied up in managing incomplete requests.

Countermeasures include setting lower timeouts for incomplete requests and using anomaly detection systems to identify and block connections exhibiting this behavior. Additionally, implementing flow control mechanisms to limit the allocation of resources for incomplete requests can help mitigate the impact of this attack. Monitoring network traffic for patterns such as repeated incomplete POST requests can also aid in early detection.

4.2.3 Attack Version 3

Sending Connection Preface Only attack exploits the initial handshake process of the HTTP/2 protocol. HTTP/2 requires clients to send a connection preface to establish a session with the server. This preface includes essential settings that enable the protocol's features, such as multiplexing and flow control. In this attack, the attacker sends only the connection preface and refrains from transmitting subsequent requests or frames.

The server, adhering to the protocol's expectations, maintains the connection and awaits further communication from the client. This behavior leads to resource allocation for each malicious connection, including memory for session management and threads for processing incoming frames. The attacker's minimal activity ensures that the attack is stealthy and consumes negligible resources on the attacker's side.

To execute this attack, the attacker establishes multiple connections with the server, sending only the connection preface for each one. The server, expecting legitimate communication, retains these connections in an active state. Over time, the accumulation of such connections can overwhelm the server's capacity, causing a denial of service.

Mitigation involves implementing connection timeouts for incomplete sessions and monitoring the rate of connections that fail to progress beyond the preface stage. Servers can also deploy heuristics

to detect and terminate connections exhibiting suspicious patterns, such as an unusually high number of connections stuck at the preface stage. Rate-limiting new connections and employing distributed denial-of-service (DDoS) protection mechanisms can further reduce the attack's impact.

4.2.4 Attack Version 4

The Incomplete GET/POST Request Header attack manipulates the HTTP/2 headers to exploit the server's processing logic. In a typical HTTP/2 session, headers are transmitted in HEADERS frames, which include flags to indicate the completion of the header section. The attacker sends a HEADERS frame with the 'END HEADERS' flag unset, signaling that additional headers will follow, even though the attacker has no intention of sending them.

The server, following the protocol's specifications, waits indefinitely for the remaining headers. This behavior ties up resources, as the server allocates memory and processing capacity to handle the incomplete request. When executed concurrently across multiple connections, this attack can exhaust the server's resources, preventing it from serving legitimate clients.

Attackers often randomize the content of the initial HEADERS frame to bypass basic signature-based detection systems. This randomness makes it challenging for traditional intrusion detection systems to identify and block the attack.

Countermeasures include setting timeouts for incomplete header transmissions and employing deep packet inspection (DPI) to detect and terminate connections with suspicious header patterns. Additionally, servers can implement stricter validation of HEADERS frames to identify and reject frames with inconsistent or missing flags. Rate-limiting new connections and deploying adaptive resource allocation algorithms can also help mitigate the attack's impact.

4.2.5 Attack Version 5

Unacknowledged SETTINGS Frame attack targets the SETTINGS frame exchange process, a fundamental aspect of HTTP/2 communication. The SETTINGS frame allows clients and servers to negotiate configuration parameters, such as stream limits and window sizes. In this attack, the attacker sends a SETTINGS frame but deliberately fails to acknowledge the server's SETTINGS frame in response.

HTTP/2 requires that each SETTINGS frame be acknowledged with a specific acknowledgment frame. By withholding this acknowledgment, the attacker disrupts the protocol's normal flow, causing the server to wait indefinitely. This behavior ties up resources allocated for the connection, as the server anticipates further communication that never arrives.

To execute this attack, the attacker initiates multiple connections with the target server, sending SETTINGS frames without acknowledgment. The server, adhering to protocol expectations, retains these connections, consuming resources for each one. Over time, the accumulation of such connections can degrade server performance or lead to a complete denial of service.

Mitigation strategies include implementing timeouts for unacknowledged SETTINGS frames and using heuristics to identify and terminate connections exhibiting this behavior. Servers can also monitor the acknowledgment patterns of SETTINGS frames to detect anomalies and block malicious connections. Deploying rate-limiting mechanisms and employing advanced anomaly detection systems can further enhance the server's resilience against this attack.

4.2.6 Summary of Findings

Our research highlights critical vulnerabilities in HTTP/2 servers that arise from the protocol's design and operational assumptions. These vulnerabilities are exacerbated by the stealthy nature of Slow Rate DoS attacks, which exploit server behavior while generating minimal network traffic. Key findings from our investigation include:

1. Vulnerability Analysis: HTTP/2 servers are susceptible to various attack strategies that exploit multiplexing, flow control, and connection management mechanisms. These attacks can effectively exhaust server resources with minimal computational effort from the attacker.

2. Attack Diversity: The distinct characteristics of each attack variant necessitate tailored mitigation approaches. For example, attacks leveraging incomplete request headers require different countermeasures than those exploiting zero window advertisements.

3. Detection Challenges: Traditional detection mechanisms, which often rely on volume-based thresholds, struggle to identify Slow Rate DoS attacks due to their low traffic footprint. This necessitates advanced approaches like event sequence analysis for real-time anomaly detection.

4. Mitigation Insights: Effective countermeasures include configuring stricter timeouts, deploying adaptive resource allocation strategies, and implementing robust anomaly detection systems. However, these measures must also account for legitimate clients with poor network conditions to avoid inadvertently denying service.

5. Real-World Implications: The heterogeneity of HTTP/2 server implementations introduces variability in vulnerability levels, emphasizing the need for consistent security practices across deployments.

Overall, our findings emphasize the need for a proactive and adaptive approach to securing HTTP/2 servers against evolving attack strategies. By understanding the nuances of these vulnerabilities, organizations can implement more effective defenses and ensure the robustness of their web infrastructure in the face of emerging threats.

5 Detection Approach

The method involves learning patterns of HTTP/2 communication based on observed packet behavior and using this knowledge to identify anomalies in traffic. The main steps of the approach involve dataset processing, learning the stream behavior, and detection based on learned behavior.

5.1 Dataset

For the learning phase of our low-and-slow DoS attack detection model, we utilized a dataset consisting of pcap files capturing normal HTTP/2 traffic. These files were sourced from real-world browsing sessions on HTTP/2-supported web servers, with no attack activity present during the capture. This data represents typical web traffic, where packets are transmitted smoothly and without malicious interruptions, providing a clear baseline of normal user behavior.

The dataset was preprocessed in the process, which involved splitting the pcap files into individual HTTP/2 streams. This ensured that each flow was analyzed independently. We then extracted relevant information, including flow identifiers, packet payloads, and timestamps, enabling a detailed examination of traffic patterns.

Since HTTP/2 traffic is typically encrypted using TLS, the pcap files were decrypted to analyze the packets at the application layer. This step allowed us to inspect connection establishment times, request-response cycles, and data flow between clients and servers. The dataset was further processed to extract event sequences, lookahead pairs, and delays between events. These metrics characterize the normal operational behavior of HTTP/2 communication, serving as a foundation for our model.

Key features, such as the timing and size of packets, the flow of HTTP/2 frames, and the dynamics of request-response interactions, were captured to model normal traffic behavior. This analysis provided the necessary insights to train our model to distinguish legitimate traffic from anomalies. Specifically, delays between consecutive events and mismatched sequences were identified as critical indicators of low-and-slow DoS attacks, guiding the development of detection algorithms.

By using this dataset of untainted HTTP/2 traffic, we established a robust understanding of normal web server behavior, enabling our system to detect deviations caused by low-and-slow DoS attacks effectively.

5.2 Solution approach

The approach is described in the following steps:

1. **Dataset Segmentation** The first step in our approach is to segment the dataset by separating individual HTTP/2 streams based on their TCP stream identifiers. This ensures that we analyze each stream independently, avoiding the interference of unrelated traffic and allowing for precise behavioral analysis of each stream.

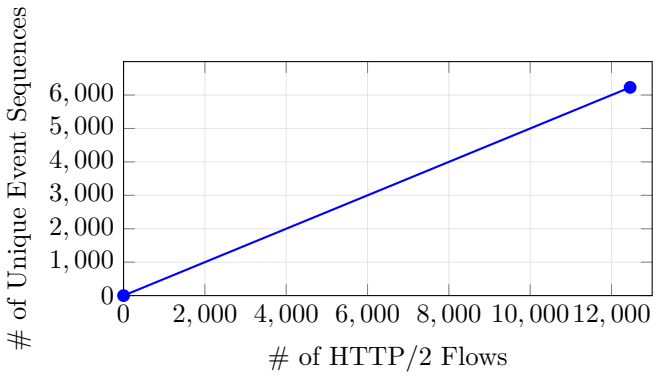


Figure 1: Relationship between HTTP/2 Flows and Unique Event Sequences

2. **Learning Stream Behavior** Once the streams are segmented, the next step is to learn the normal behavior of each stream. This learning process involves extracting sequences of events, such as packet arrivals or responses, and calculating the time delays between them. Then store these event pairs, along with their corresponding delays, forming a database of normal stream behaviors. These event pairs, or lookahead pairs, serve as a basis for future comparisons during the detection phase.

3. **Learning Phase** The learning phase focuses on extracting the behavior of each HTTP/2 stream by observing the event sequences and their delays over the course of traffic communication. During this phase, the algorithm calculate delay thresholds for each stream based on the observed behavior, which will later serve as reference values for the detection phase.

4. **Detection Phase** After learning the normal patterns of HTTP/2 streams, move to the detection phase, where it analyzes incoming traffic for any deviations from the learned patterns. This phase involves parsing each event in the traffic and comparing it with the learned lookahead pairs and delay thresholds. If any event or delay does not match the expected behavior, it is flagged as a potential anomaly, indicating a possible low-and-slow attack.

   The steps involved in the detection phase are as follows:

   - Event Parsing: For each stream in the incoming traffic, we parse the event sequences and their corresponding timestamps.
   - Event Comparison: For each event e_t in the sequence, we retrieve the next event e_{t+1} and calculate the delay Δt.
   - Anomaly Detection: If the pair (e_t, e_{t+1}) does not exist in the learned set of lookahead pairs L or the computed delay Δt exceeds the pre-defined threshold, it is flagged as a delay violation.
   - Anomaly Logging: Any detected anomalies are logged for further analysis, and an anomaly counter is incremented to track the number of violations.

   The pseudocode for this detection process is shown in Algorithm 1.

5. **Reporting and Anomaly Detection Results** After the detection phase, the system outputs a report summarizing the detected anomalies, including details on the streams exhibiting suspicious delays or mismatches. These reports can be further analyzed by system administrators or security personnel to determine if a low-and-slow DoS attack is occurring.

This approach combines behavioral learning and real-time detection to identify low-and-slow DoS attacks in HTTP/2 traffic. By focusing on the patterns and delays inherent in the communication streams, our method is able to detect deviations from normal traffic behavior, thereby providing an effective means of identifying malicious activities targeting HTTP/2 services.

5.3 Learning Phase

In the learning phase of our low-and-slow DoS attack detection model, we analyzed a dataset of real-world HTTP/2 traffic consisting of 17,694 streams captured from routine browsing sessions. Each TCP stream represented a unique client-server interaction and was processed independently to ensure precise analysis. The streams were parsed to extract critical details, including flow identifiers (source and destination IPs and ports), packet payloads, and timestamps.

The extracted data was used to model each flow as a sequence of events. Each flow began with a Start marker, indicating the initiation of the connection, and ended with an End marker, signaling its termination. In between these markers, HTTP/2 frames were mapped to specific events, such as Headers, Data, Ping, or Goaway, depending on their type and behavior. These event sequences captured the complete lifecycle of an HTTP/2 connection, providing a detailed representation of its structure and dynamics. The Start and End markers not only defined clear boundaries for analysis but also ensured that each sequence could be processed as a self-contained unit, facilitating accurate modeling of the flow's behavior.

Beyond constructing sequences, we analyzed the timing between consecutive events to capture temporal patterns. For each event transition, we calculated the time delay and recorded the maximum observed delay in a database. This database of delays formed a critical baseline for identifying anomalies during the detection phase, as it highlighted the typical timing relationships between events in legitimate traffic.

To further understand dependencies within the traffic, we extracted relationships between events, known as lookahead pairs. Each event was linked to subsequent events within a defined range (e.g., three to seven events ahead). These pairs captured patterns of how events in HTTP/2 traffic are related, such as the order in which frames appear and their relative proximity. For example, a sequence might reveal that Headers frames are often followed by Data frames within a specific time window, creating predictable patterns in normal traffic. These pairs were stored in a database, which provided an efficient summary of typical event dependencies.

The results of this learning phase were exported into structured files for use during the detection phase. Event sequences were stored as ordered records to preserve their structural integrity, while delay information and lookahead pairs were organized into dedicated files. This allowed the detection system to quickly compare observed traffic against the learned models of normal HTTP/2 behavior.

By combining detailed sequence analysis, timing patterns, and event relationships, the learning phase established a comprehensive understanding of legitimate HTTP/2 communication. The inclusion of Start and End markers defined precise boundaries for each flow, while the extracted delays and lookahead pairs provided insights into the traffic's internal structure and timing. This robust baseline enabled our system to detect deviations caused by low-and-slow DoS attacks with high precision and scalability.

5.4 Detection Phase

The detection phase of our low-and-slow DoS attack detection model focused on analyzing HTTP/2 traffic to identify anomalous behaviors. The traffic was processed by reconstructing event sequences, analyzing delays between events, and comparing these observations against the previously learned baseline of normal HTTP/2 behavior. For each incoming TCP stream, packets were parsed to extract key information, including flow identifiers, timestamps, and packet payloads. Each packet was mapped to an event, such as Headers, Data, or Ping, based on the HTTP/2 frame type and its associated flags. These events were concatenated into sequences, preserving the order of occurrences within the flow, which provided a comprehensive view of the connection's behavior.

To identify potential anomalies, the detection system calculated mismatch scores by comparing the observed event sequences to the database of learned lookahead pairs. Each pair in the database represented the expected relationship between events within a specified range. Deviations, such as missing or unexpected pairs, contributed to the mismatch score for the sequence. A sequence was flagged as anomalous if its mismatch score exceeded a predefined threshold.

In addition to mismatch scores, the system analyzed timing information between consecutive events. For each pair of events in a sequence, the actual time delay was computed and compared against the maximum allowable delay stored in the learned delay database. Any delay exceeding this threshold

was flagged as a potential anomaly. These delay violations were logged for further analysis, providing additional evidence of unusual traffic behavior.

Throughout the detection phase, the system maintained detailed logs of its analysis. Event sequences were recorded to monitor the behavior of each flow, including normal and anomalous patterns. Delay violations and their corresponding event transitions were also written to files, enabling a deeper investigation into timing-based anomalies. Furthermore, the system documented mismatch scores and flagged sequences, offering insights into how deviations from normal traffic patterns contributed to the detection process.

By combining sequence analysis, timing verification, and detailed logging, the detection phase provided a robust and scalable method for identifying low-and-slow DoS attacks. The logs generated during this phase not only facilitated real-time anomaly detection but also supported post-event analysis to refine detection strategies and better understand attack patterns.

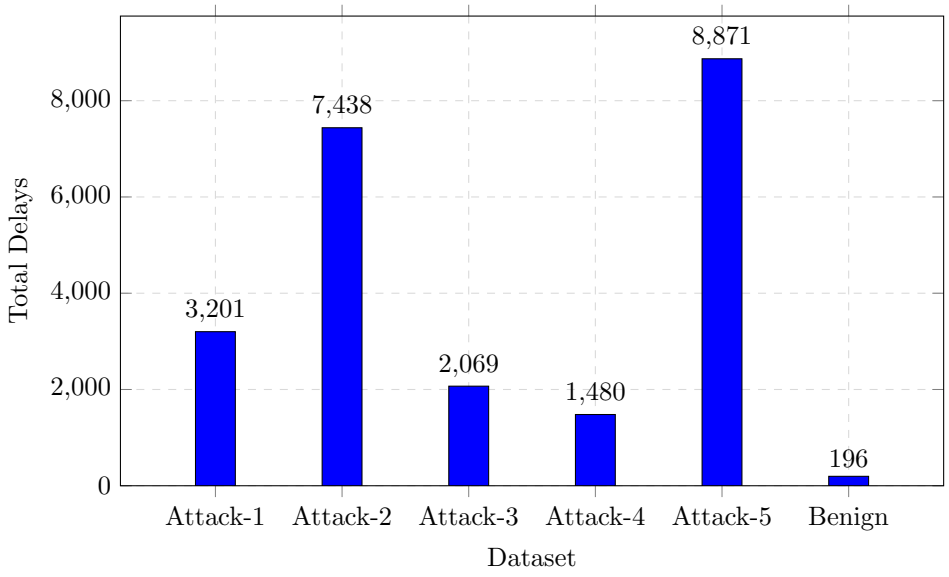


Figure 2: Proportion of delays across datasets (Attack 1–5 and Benign).

Algorithm 1 Detection Phase

Input: Lookahead dictionary $D_{\text{lookahead}}$, delay thresholds D_{delay}, number of events n, current time t, list of packets *packets*, and pcap file name *pcap_name*
Output: Detected delays and anomalies
Initialize *event_sequence* ← "", *detected_events* ← [], *events_time* ← []
Initialize *current_event_index* ← 0, *delay_counter* ← 0
for all *packet* **in** *packets* **do**
    frame_data ← *packet*[*data*]
    event ← FindEvent(*packet*[*flowID*], *frame_data*, **None**, *packet*[*time*], 0)
    if *event* ≠ **None then**
        current_event_index ← *current_event_index* + 1
        Append *event* to *event_sequence*
        Append *packet*[*time*] to *events_time*
        Append *event* to *detected_events*
        FindEventTiming(*packet*[*flowID*], *event*, *packet*[*time*])
    end if
end for
new_events ← Split(*event_sequence*, "-¿*")
if Length(*new_events*) > 1 **then**
    total_events ← Length(*new_events*)
    for *event_index* ← 1 **to** *total_events* − 1 **do**
        event_transition_key ← *new_events*[*event_index* - 1] + *new_events*[*event_index*]
        WriteToFile("DelaysDetect", *event_transition_key*, "a")
        if *event_transition_key* ∈ D_{delay} **then**
            max_delay ← D_{delay}[*event_transition_key*]
            if *max_delay* ≠ 0 **then**
                actual_delay ← *events_time*[−1] − *events_time*[−2]
                if *actual_delay* > *max_delay* **then**
                    WriteToFile("DelaysDetect_{*pcap_name*}",
                        "Attack: Delay detected - Transition: {*event_transition_key*}, Max: {*max_delay*}, Actual: {*actual_delay*}", "a")
                    delay_counter ← *delay_counter* + 1
                end if
            end if
        end if
    end for
end if
WriteToFile("DelaysDetect_{*pcap_name*}", "Found {*delay_counter*} delays", "a")
Output: Detected delay violations and total delays

5.5 Core Anomaly Detection in HTTP/2 Streams

The cornerstone of our anomaly detection framework is the **Mismatch Detection** mechanism, which systematically identifies deviations between HTTP/2 traffic behavior and established normal patterns learned during the training phase. This detection strategy operates by meticulously analyzing incoming HTTP/2 event sequences and the temporal delays between these events, leveraging the functions

Initially, the system undergoes a **learning phase**, where it processes captured network traffic to construct a comprehensive model of typical HTTP/2 stream behavior. This involves parsing PCAP to extract relevant packet information, which filters out non-HTTP/2 traffic and aggregates packets based on unique flow identifiers. The learning function then generates **lookahead pairs** through extract LookaheadPairs, which examines sequences of HTTP/2 events within a specified window size to capture the permissible transitions and their corresponding temporal characteristics. These lookahead pairs are stored in the Dlookahead dictionary, encapsulating the learned event transition patterns essential for future comparisons.

Concurrently, the system calculates **delay thresholds** between consecutive events using the calculate Avg Delay Between Events . This process involves computing the maximum observed delays

for each event transition, which are subsequently stored in the Ddelay dictionary. These thresholds serve as critical benchmarks; any incoming event sequence exhibiting delays exceeding these values is flagged for further scrutiny.

During the **detection phase**, incoming HTTP/2 streams are continuously monitored and evaluated against the learned models. The detection phase and detection phase mismatch are pivotal in this stage, systematically verifying two primary conditions to ascertain anomalies:

1. **Lookahead Pair Mismatch**: Utilizing the precomputed Dlookahead dictionary, the system checks whether each observed event pair (e_t, e_{t+1}) exists within the set of learned lookahead pairs. The detection phase mismatch employs the extract Lookahead Pairs to parse the current event sequence and identify any unexpected transitions, thereby signaling potential anomalies such as low-and-slow Denial-of-Service (DoS) attacks that manipulate event sequences to evade conventional detection.

2. **Delay Threshold Violation**: The system assesses the temporal delays between consecutive events against the established thresholds in Ddelay. If the delay Δt for any event transition surpasses its corresponding threshold, the event sequence is marked as suspicious. This is particularly effective in identifying low-and-slow attacks, where attackers intentionally introduce delays to blend malicious traffic with normal traffic patterns.

The integration of these two detection criteria enhances the validation of the anomaly detection system, enabling it to discern subtle deviations that may indicate sophisticated attacks. Furthermore, the implementation includes comprehensive logging and visualization capabilities, as evidenced by the plotting functions which generate graphical representations of event delays and mismatch occurrences. These visualizations facilitate an in-depth analysis of traffic patterns and the effectiveness of the detection mechanisms.

Empirical evaluations demonstrate that this mismatch-based detection approach effectively identifies anomalies characterized by irregular event sequences and atypical delays, thereby providing a resilient defense against evasive low-and-slow DoS attacks. The modularity of the codebase, with clearly defined functions for each phase of detection, ensures scalability and adaptability to evolving HTTP/2 traffic behaviors, making it a viable solution for real-time network security monitoring.

Algorithm 2 Mismatch Detection Phase

Input: Lookahead dictionary $D_{\text{lookahead}}$, delay thresholds D_{delay}, lookahead window size n, threshold t, list of packets *packets*, and pcap file name *pcap_name*
Output: Detected mismatch events and anomalies
Initialize *detected_events* $\leftarrow$ [], *timeout_counter* $\leftarrow 0$
for all *packet* **in** *packets* **do**
    frame_data $\leftarrow$ *packet*[*data*]
    event $\leftarrow$ FindEvent(*packet*[*flowID*], *frame_data*, **None**, *packet*[*time*], 0)
    if *event* $\neq$ **None then**
        Append *event* to *seq_de*
        Append *event* to *detected_events*
        FindEventTiming(*packet*[*flowID*], *event*, *packet*[*time*])
    end if
end for
if *detected_events* is empty **then**
    return
end if
last_event $\leftarrow$ *detected_events*[-1]
max_delay $\leftarrow \max(D_{\text{delay}}[\textit{last\_event}], \{*\} : 0)$
last_event_time $\leftarrow$ *event_timings*[*packet*[*flowID*]][-1]
if *last_event_time* $\neq$ **None and** (time.now() $-$ *last_event_time*) $>$ *max_delay* **then**
    Append "$\rightarrow$ TO*timeout_counter* $\rightarrow$ *" to *seq_de*
    WriteToFile("DetectMissmatch_*pcap_name*", "Delay detected - *last_event*", "a")
    timeout_counter $\leftarrow$ *timeout_counter* $+ 1$
end if
mismatch $\leftarrow 0$
if Length(*detected_events*) $>$ n **then**
    lookahead_pairs $\leftarrow$ ExtractLookaheadPairs(*seq_de*, n)
    for all *pair* **in** *lookahead_pairs* **do**
        if *pair* $\notin D_{\text{lookahead}}$ **then**
            mismatch $\leftarrow$ *mismatch* $+ 1$
            WriteToFile("DetectMissmatch_*pcap_name*", "Added to mismatch", "a")
        end if
    end for
end if
mismatch_ratio $\leftarrow \frac{\textit{mismatch}}{n \cdot \left(\text{Length}(\textit{detected\_events}) - \frac{n+1}{2}\right)}$
if *mismatch_ratio* $>$ t **then**
    WriteToFile("DetectMissmatch_*pcap_name*", "Sequence is anomalous", "a")
    detect_count $\leftarrow$ *detect_count* $+ 1$
else if *mismatch_ratio* $<$ t **and** *detected_events*[-1] = "$\rightarrow$ Goaway $\rightarrow$ *" **then**
    WriteToFile("DetectMissmatch_*pcap_name*", "Sequence is normal", "a")
end if

5.6 Detection Phase Mismatch Procedure

1. **Event Parsing:** In line with the detection procedure, the first step is to parse each event in the HTTP/2 stream and retrieve its corresponding timestamp. This allows the system to assess the temporal behavior of the traffic.

2. **Event Pair and Delay Comparison:** For each event e_t, the algorithm retrieves the subsequent event e_{t+1} and calculates the delay Δt between the two events. The delay is then compared against the predefined threshold.

3. **Mismatch Condition:** The core operation of the algorithm checks whether the pair (e_t, e_{t+1}) exists in the learned set of look-ahead pairs L. If the pair is not present or if the delay Δt exceeds the threshold, a mismatch is detected.

4. **Anomaly Logging:** Each mismatch is logged, and the mismatch counter is incremented. This counter serves as a measure of the frequency and severity of the deviations, allowing for a quantitative assessment of traffic anomalies.

5. **Final Output:** After processing all HTTP/2 streams, the system generates a comprehensive report of the detected mismatches and anomalies. This report is essential for further analysis, enabling the identification of potential security threats or performance issues in the traffic.

The Mismatch Detection phase is central to identifying low-and-slow DoS attacks, which are characterized by subtle deviations in traffic behavior, often involving delays rather than large-scale packet flooding. By examining event sequences and delay violations, this phase provides a methodical approach to detect anomalies that may otherwise be difficult to uncover. This mismatch-based detection approach offers an effective tool for identifying sophisticated attacks that exploit delays in HTTP/2 communication, thereby enhancing the security of web traffic systems.

6 Evaluation

To evaluate the effectiveness of our proposed detection mechanism, we conducted extensive experiments on both legitimate HTTP/2 browsing traffic and simulated low-and-slow attack scenarios. The evaluation was designed to assess the system's ability to accurately detect anomalies in individual HTTP/2 streams while maintaining efficiency in handling high volumes of traffic.

6.1 Experimental Setup

The dataset used for this evaluation comprised two distinct components:

- **Legitimate Traffic**: HTTP/2 traffic was collected during normal browsing sessions across various popular websites. This dataset was used to establish a baseline of typical HTTP/2 behaviors, including event sequences and timing patterns.

- **Attack Traffic**: Simulated low-and-slow attacks were generated by crafting malicious HTTP/2 requests that exploited delays, incomplete data transmissions, and anomalous event sequences.

The PCAP files for both datasets were processed using our detection framework. Each file was split into individual HTTP/2 streams, ensuring granular analysis of the behaviors associated with each connection. This segmentation allowed us to study traffic characteristics at the stream level without interference from unrelated data.

Our experimental environment consisted of a standard server-client architecture where HTTP/2 traffic was captured using Wireshark. The server's performance was monitored under normal and attack conditions, focusing on resource utilization and response times.

6.2 Methodology

The evaluation process followed these key steps:

1. **Stream Segmentation**:

   - Each PCAP file was divided into individual streams.
   - Streams were identified by unique TCP flow identifiers, ensuring isolation of traffic behaviors.

2. **Baseline Learning**:

   - The system processed legitimate traffic streams to extract event sequences and calculate delays between consecutive events.
   - Lookahead pairs and delay thresholds were established as benchmarks for normal traffic behavior.

3. **Anomaly Detection**:

- For each stream in the attack PCAP files, the system compared observed event sequences against the learned baseline.
- Anomalies were flagged based on two criteria:
  (a) **Lookahead Pair Mismatches**: Event pairs not present in the baseline lookahead pairs were identified as potential anomalies.
  (b) **Delay Threshold Violations**: Delays between consecutive events exceeding the learned thresholds were flagged as suspicious.

4. **Performance**:

	HTTP2 Streams	**Delays**	**Anomalies**
Attack 1	3318	3198	3201
Attack 2	3311	3199	7438
Attack 3	3318	3184	2069
Attack 4	3305	3150	1480
Attack 5	3303	3166	8871

Table 1: Attacks HTTP2 Streams Analysis Result

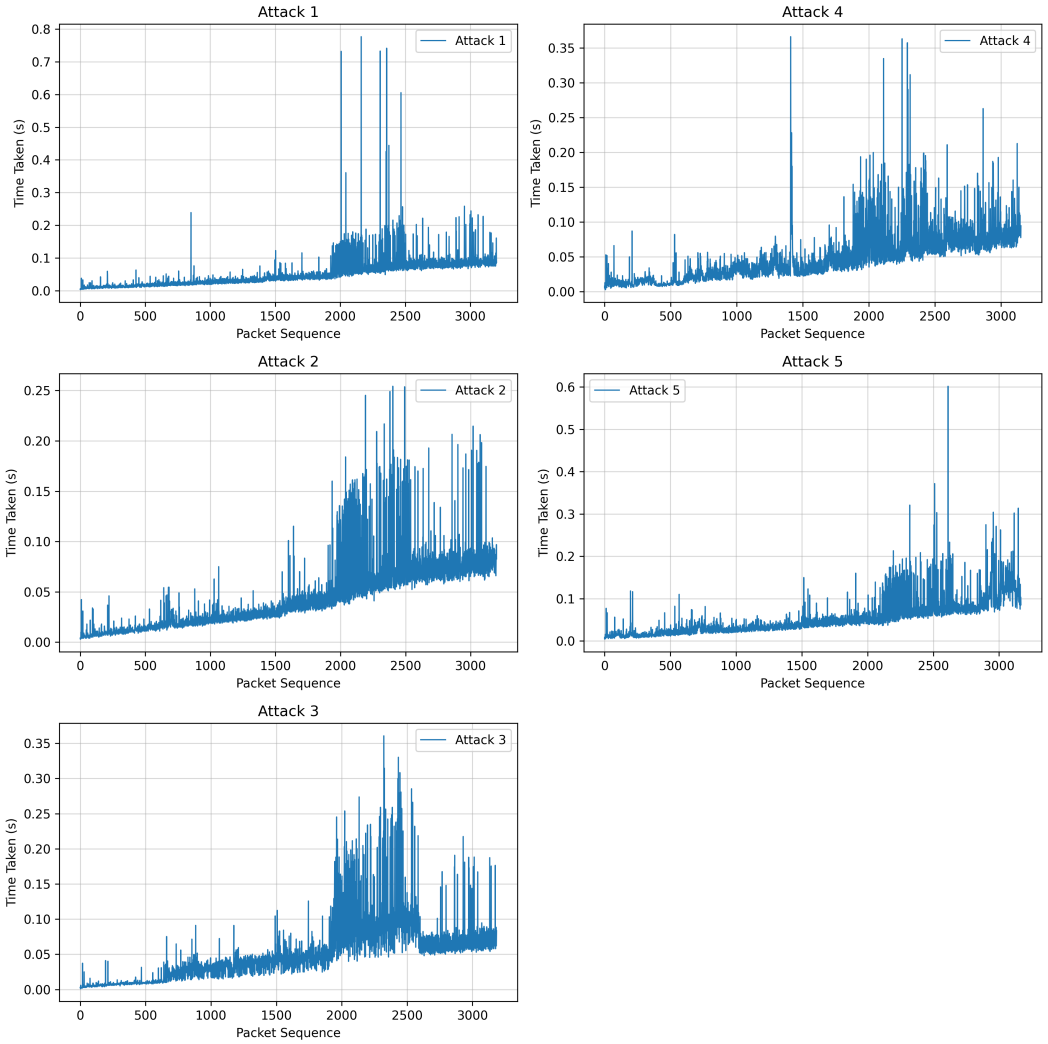


Figure 3: Attack Time Detection

6.3 Results

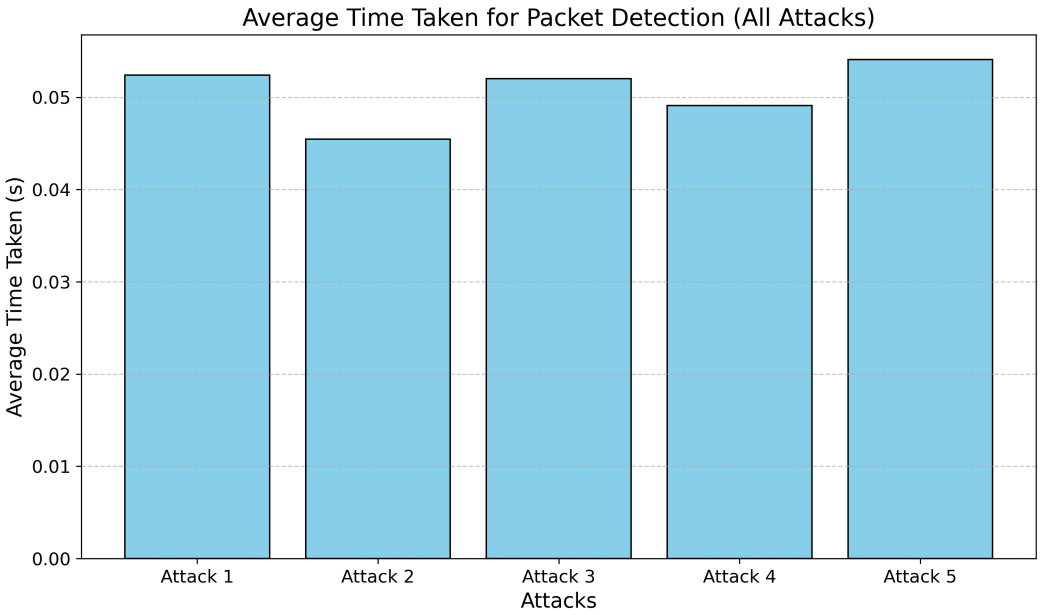
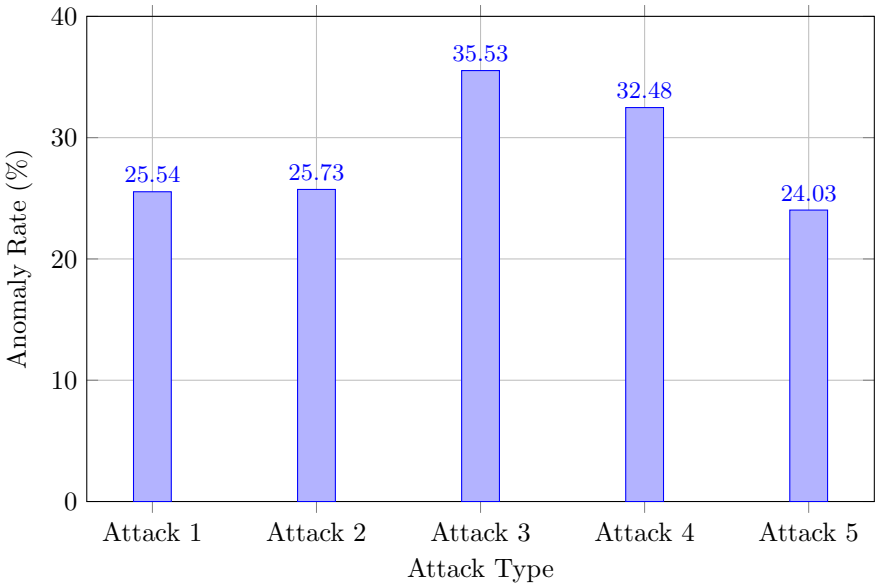


Figure 4: Average Attack Time Detection



In the evaluation, we observe the effectiveness and reliability of the model in accurately detecting anomalies while also providing noticeable performance in determining whether the traffic is anomalous or benign. The model successfully handles various types of Low and Slow HTTP/2 attacks with impressive speed and precision.

- **Detection Accuracy**: The system demonstrated exceptional performance in identifying anomalies across nearly all attack streams, achieving a consistently high detection rate across diverse scenarios and traffic patterns.
- **False Positives**: When tested on legitimate browsing traffic, the system effectively distinguished between benign and malicious behaviors, showcasing a low rate of false positives and a strong ability to differentiate anomalies from normal operations.

- **Efficiency**: The model efficiently processed approximately 10,000 packets within just a few seconds, highlighting its scalability and suitability for real-time deployment in high-traffic environments.

6.4 Detailed Analysis

1. **Attack Traffic Analysis**:
   - In simulated attack scenarios, the system flagged streams with anomalous delays and unexpected event sequences.
   - The detection mechanism's reliance on both event sequence mismatches and timing irregularities ensured comprehensive coverage of diverse attack patterns.
2. **Legitimate Traffic Analysis**:
   - The system accurately classified streams from normal browsing sessions as non-anomalous, with minimal false positives. This highlights the robustness of the baseline learning phase in capturing the variability of legitimate HTTP/2 traffic.
3. **Visualization and Logging**:
   - The generated logs and visualizations, histograms of mismatches, provided valuable insights into traffic behaviors and anomalies. These outputs facilitate post-event analysis and refinement of detection strategies.

6.5 Discussion

The results underscore the effectiveness of our approach in detecting low-and-slow attacks by isolating and analyzing individual HTTP/2 streams. Key takeaways include:

- **Stream-Level Analysis**: Segmenting PCAP files into streams proved essential for accurately profiling traffic behaviors. This granular approach minimized interference from unrelated flows and enabled precise anomaly detection.
- **Robust Detection Criteria**: Combining lookahead pair mismatches and delay threshold violations enhanced the system's ability to detect both subtle and overt attack behaviors.
- **Scalability**: The system's ability to handle high packet processing rates with minimal computational overhead makes it suitable for real-world deployment.

7 Conclusion

Low-and-slow denial-of-service (DoS) attacks present a critical challenge for HTTP/2 servers due to their ability to evade traditional detection mechanisms. These attacks exploit the flow control and multiplexing features of HTTP/2 to send minimal amounts of data at slow rates, making them highly stealthy and difficult to identify. Existing countermeasures designed for HTTP/1.1 are insufficient to address the unique characteristics of HTTP/2, highlighting the need for specialized detection techniques.

In this work, we proposed an approach to detect low-and-slow DoS attacks on HTTP/2 traffic by analyzing event sequences and delay patterns in network streams. The proposed method consists of a learning phase that extracts lookahead pairs and delay thresholds from normal traffic to establish a baseline for expected behavior. In the subsequent detection phase and mismatch detection phase, the system identifies deviations by comparing incoming traffic against the learned patterns. Specifically, mismatched event sequences and excessive delays between events are flagged as anomalies indicative of potential low-and-slow DoS attacks.

Our experimental results demonstrate that the proposed detection mechanism can accurately identify low-and-slow attacks, even in highly stealthy scenarios. By focusing on the temporal behavior of HTTP/2 streams and analyzing event mismatches, the approach effectively differentiates malicious traffic from legitimate HTTP/2 interactions. Furthermore, the computational overhead introduced by

the detection process is minimal, making the solution suitable for practical deployment in real-world environments.

Acknowledgments We acknowledge the Ariel Cybersecurity Center at Ariel University for providing computing resources that have contributed to the research results reported within this paper.

References

[1] Nagy, R. & Ihab, A. (2016). *Impact of Implementing HTTP/2 in Web Services.* In International Journal of Computer Applications.

[2] S. Mansfield-Devine, "DDoS: threats and mitigation," Network Security, vol. 2011, pp. 5-12, 2011.

[3] S. Heron, "Denial of service: motivations and trends," Network Security, vol. 2010, pp. 10-12, 2010.

[4] E. Adi, "Low-Rate Denial-of-Service Attacks against HTTP/2 Services," In 2015 5th ICITCS.

[5] K. Sonar, "A Survey: DDOS Attack on Internet of Things," IJERD, vol. 2010, pp. 58-63, 2014.

[6] G. Fernanzed, "Evaluation of a low-rate DoS attack against application servers," Computers & Security, vol. 27, pp. 335-354, 2008.

[7] V. De Miranda Rios, "Detection and Mitigation of Low-Rate Denial-of-Service Attacks: A Survey," IEEE, 2022.

[8] R. Malhotra, "Exploiting HTTP/2 Protocol Features for Low-Rate Denial-of-Service Attacks," Proceedings of the ACM International Conference on Web Security, 2017.

[9] Z. Zhao, "Security Risks and Attack Strategies in HTTP/2: A Study of Low-Rate DoS Attacks," IEEE Transactions on Network and Service Management, 2019.

[10] S. Zhang, "Detection of Low-Rate Denial of Service Attacks in HTTP/2 Traffic," Computer Communications, 2020.

[11] F. Yang, "Anomaly-Based Detection of Low-Rate DoS Attacks in HTTP Traffic," Proceedings of the 2018 IEEE International Conference on Communications (ICC), 2018.

[12] J. Huang, "Real-Time Detection of Low-Rate Denial of Service Attacks Using Statistical Analysis," Journal of Network and Computer Applications, 2019.

[13] N. Tripathi, "Slow Rate Denial of Service Attacks Against HTTP/2 and Detection," Computers & Security, vol. 72, pp. 255-272, 2018.

[14] N. Tripathi, "Delays have Dangerous Ends: Slow HTTP/2 DoS attacks into the Wild and their Real-Time Detection using Event Sequence Analysis," IEEE, 2022.

[15] C. Lee, "PRETT2: Discovering HTTP/2 DoS Vulnerabilities via Protocol Reverse Engineering," Springer Nature, pp. 3-23, 2024.