The ISC Int'l Journal of Information Security
ISeCure

January 2024, Volume 16, Number 1 (pp. 79–92)
http://www.isecure-journal.org

Slowloris Attack Detection Using Adaptive Timeout-Based Approach

Kangkan Talukdar [1] and Debojit Boro [2,*]
[1] *Numaligarh Refinery Limited, Numaligarh, Assam, India*
[2] *Department of Computer Science and Engineering, Tezpur University, Assam, India*

ARTICLE INFO.

Article history:
Received: April 9, 2023
Revised: July 21, 2023
Accepted: August 7, 2023
Published Online: November 20, 2023

Keywords:
Adaptive Timeout, DDoS Attack, Flooding attack, HTTP Protocol, Slowloris Attack

Type: Research Article

doi: 10.22042/isecure.2023.392462.938

dor: 20.1001.1.20082045.2024.16.1.5.9

ABSTRACT

Distributed Denial of Service (DDoS) attacks have become a critical threat to the Web with the increase in web-based transactions and application services offered by the Internet. With the vast resources and techniques readily available to the attackers, countering them has become more challenging. They are usually carried out at the network layer. Unlike traditional network-layer attacks, application-layer DDoS attacks can be more effective. It utilizes legitimate HTTP requests to inundate victim resources that are undetectable. Many methods exist in the literature to protect systems from IP and TCP layer DDoS attacks that do not work when encountering application-layer DDoS attacks. Most network-layer DDoS attacks are flooding attacks, but application-layer DDoS attacks can be flooding or protocol-specific vulnerability attacks. Various protocol-specific vulnerability attacks cannot be detected by traditional detection methods as they are designed to detect flooding attacks. One such attack is the slowloris attack. It targets web servers by exploiting an HTTP protocol vulnerability. In this paper, we propose a slowloris attack detection based on an adaptive timeout-based approach that contains two modules: a suspect determination module and an attacker verification module. The determination module determines suspects and sends them to the verification module, which verifies a suspect as an attacker. We have designed a detection algorithm that detects an attacker's IP address before it consumes all the resources. The experimental results substantiate its efficacy with low false alarms and high detection accuracy.

1 Introduction

A DDoS attack is a large-scale coordinated attack on the available services of a victim system or network resource that is launched indirectly through many compromised computers on the Internet [1]. They have been known to the network research community since the early 1980s, with its first incident reported in the summer of 1999 by the Computer Incident Advisory Capability (CIAC) [2]. They aim at any network device but most often at application layer servers, like electronic mail servers, web servers, DNS servers, etc., to render the most popular services unavailable for users. This is mainly done by

* Corresponding author.
Email addresses: ktalukdar.dbcet@gmail.com, deb0001@tezu.ernet.in
ISSN: 2008-2045

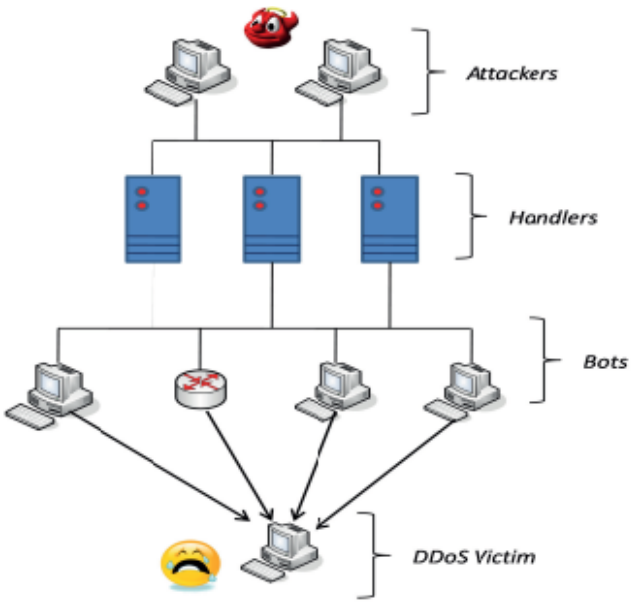

Figure 1. Components of Botnet-based DDoS attack

consuming the network bandwidth, the victim's device RAM, or the CPU cycles [3]. DDoS attackers first set up a botnet by infiltrating large numbers of computers by exploiting their software vulnerabilities. These compromised computers in the botnet are then orchestrated to wage a large-scale coordinated attack against one or more victim systems [4]. As shown in Figure 1, the attacker compromises a first tier of vulnerable computers known as handlers. The attacker then orders a second tier of several more computers through them to simultaneously attack a specific target. Based on the OSI layer a DDoS attack targets and the attack traffic flow rate during the attack, a DDoS attack can be classified into two types. If the attack targets the application layer, then it is categorized as an App-DDoS attack, and if it targets the network layer, then it is categorized as a Net-DDoS attack.

Net-DDoS and Transport layer DDoS attacks are mainly flooding attacks that send lots of unwanted traffic to the target network and disrupt legitimate users' connectivity by exhausting the victim network's bandwidth. They are mainly launched using spoofed or non-spoofed TCP, ICMP, and UDP packets. An attacker exploits specific features or implementation bugs of some of these protocols to consume excess amounts of the victim's resources (e.g., TCP SYN flood, TCP SYN-ACK flood, ACK & PUSH ACK flood, etc.[5–8]. Instead of sending direct requests to the reflectors, attackers also send forged requests (e.g., ICMP echo requests) with spoofed IP addresses. The reflectors then send their replies to the victim and exhaust their resources (e.g., Smurf and Fraggle attacks) [6, 7]. Botnets are used for both the reflection and amplification of these attacks.

An App-DDoS attack can be a flooding attack or a protocol-specific vulnerability attack. In a flooding attack, the attacker sends a lot of application traffic, such as HTTP flood, to the victim network or server masquerading as flash crowds. These attacks focus on disrupting legitimate users' services by exhausting the server resources (e.g., Sockets, CPU, memory, disk/database bandwidth, and I/O bandwidth) [9]. There are different types of App-DDoS flooding attacks, such as DNS reflection/amplification attacks [2], Session Initiation Protocol (SIP) attacks using fabricated VoIP requests from a wide range of IP addresses in a short period [5, 6, 8], HTTP GET/POST flooding attacks under session, and request flooding attack initiates single or multiple sessions with a high number of requests. In a protocol-specific vulnerability attack, App-DDoS attacks generally are low-rate, stealthier, and consume less bandwidth, unlike volumetric attacks, since they share similar features of benign traffic. However, they usually have the same impact on the services since they target specific characteristics of applications such as HTTP, DNS, or Session Initiation Protocol (SIP). These attacks include asymmetric and slow request/response attacks where the attackers send sessions with high-workload requests. For instance, in asymmetric attacks, multiple HTTP requests are embedded within a single packet and issued irregularly within a single HTTP session.

The slow request/response attacks include various attacks. For instance, the attacker sends partial HTTP requests and an incomplete set of request headers in a slowloris attack [10, 11]. These requests rapidly grow, slowly update, and never close. The attack continues until all available sockets are occupied by these requests thereby making the web server inaccessible. In an HTTP fragmentation attack, the attacker sends tiny fragments of HTTP packets very slowly until the server timeout allows. The attacker can silently bring down a web server with just a handful of bots by opening multiple sessions on each bot [8]. In a slowpost attack (a.k.a, slow request bodies or R-U-Dead-Yet (RUDY) attack) [12], the attacker first sends a complete HTTP header that defines the "content-length" field of the post message body as it sends this request for benign traffic. Then, it sends HTTP post commands slowly to send the data to fill the message body at a rate of one byte every two minutes. Hence, the server waits for each message body to complete while a slowpost attack proliferates which can bring down the web server. In a slow reading or slow response attack [13], the attacker slowly reads the responses from the server instead of slowly sending the requests. The attacker

sets a smaller receive window size greater than the target server's send buffer. Even if there is no data communication, the TCP protocol maintains open connections. The attacker exploits this to force the server to keep many open connections and eventually cause a DDoS attack on the server.

With the increase in application and business services, many application servers and network facilities suffer from DDoS attacks. Traditional DDoS attack defense systems become ineffective for App-DDoS attacks as they are mainly designed to detect Net-DDoS or transport layer DDoS attacks. For example, the "Mydoom" worm virus launched an HTTP request flooding attack on the SCO Group website in 2004 [14]. The web server crashed soon because its defense mechanisms were based on statistics of IP and TCP layers that could not differentiate between normal users and Mydoom HTTP requests. This calls for a thorough investigation of the working mechanisms of these attacks, the evolving manner of these mechanisms, and know-how on defending servers and network systems against these attacks. As the occurrence of App-DDoS flooding attacks is quite apparent, we focus on the low-rate and stealthy protocol-specific vulnerability App-DDoS attacks. Therefore, we endeavor to investigate the slowloris attack and aim to propose a solution. In this paper, we propose a slowloris attack detection based on an adaptive timeout-based approach that contains two modules: a suspect determination module and an attacker verification module. The determination module determines suspects and sends them to the verification module, which verifies a suspect as an attacker. The detection algorithm detects an attacker's IP address before it consumes all the resources with low false alarms and high detection accuracy. The contribution of the paper can be summarized as follows.

- An effective method for the detection of a slowloris attack based on an anomaly-based suspect determination algorithm and an adaptive timeout-based suspect verification module.
- A novel dataset containing normal web traffic and attack traffic generated from our University Web server.
- Validation of the method with the generated dataset with different scenarios and establish the effectiveness of the method with low false alarms and high detection accuracy.

The rest of the paper is organized as follows: Section 2 discusses the slowloris attack and how it exploits the HTTP protocol's vulnerability. Section 3 discusses current detection methods of slowloris attacks. Section 4 highlights the motivation. Section 5 presents the problem definition and assumptions. Section 6 presents our proposed approach for the detection of slowloris attacks. Section 7 reports the experimental results. Finally, Section 8 highlights the conclusion and future work.

2 HTTP Vulnerability Exploitation by the Slowloris Attack

HTTP has a mechanism to serve a request from a user whose internet connection is slow. Consequently, the web server may not receive the entire header at a time, and the header comes in parts. Therefore, the server must wait to get the entire header, and whenever the server receives an incomplete header, it will wait to receive the next part until timeout. The default timeout value of 300s is modifiable in the Apache web server. This is very useful if a website serves large files for download through HTTP. The timeout value maintains an active HTTP connection of a slow client without breaking the download. Thus, the timeout counter is reset to 1 every time the client sends some more data.

Slowloris is a GET method-based attack that exploits the above HTTP's vulnerability to bring down a web server using a single or a few machines. It does not flood the victim but uses time-delayed HTTP GET headers with spoofed requests. The attacker intentionally sends partial HTTP requests (mostly non-spoofed source IP addresses) to the server and sends bogus data very frequently to reset the timeout counter of each request [15]. These requests are sent over the full and legitimate TCP connections. The attacker sends separated lines of the header one by one and does not send an HTTP GET request header simultaneously. This causes the server to wait until the end of the request header, thus reserving the TCP connection for a long period. The default threshold of 300s in the Apache web server indicates the maximum timeout when the next line of header arrives, otherwise, the connection is closed. On the attacker side, this time is set as a break time for sending the next line of the request header. With multiple connections created in this manner, an attacker can then take up all available sockets and exhaust web server resources, thereby rendering it inaccessible. A complete HTTP GET request header looks like the below:

GET / HTTP/1.0[CRLF]
User-Agent: Wget/1.10.2 (Red Hat modified)[CRLF]
Accept: */*[CRLF]
Host: 192.168.3.18[CRLF]
Connection: Keep-Alive[CRLF][CRLF]

In the above header, CRLF stands for CR (Carriage Return) and LF (Line Feed) and are non-printable characters that denote the end of the line (EOL). Two consecutive CRLF characters represent the comple-

tion of the header and denote a blank line as in the header's "Connection" field. Slowloris exploits this in implementing its attack. It sends an incomplete request using the slowloris script and does not send a finishing blank line, as shown in the snippet below.

"GET /$rand HTTP/1.1\r\n"
. "Host: $sendhost\r\n"
. "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.503l3; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; MSOffice 12)\r\n"
. "Content-Length: 42\r\n";

"\r\n" denotes CR and LF in PERL. Two consecutive "\r\n" denote a blank line, which is not present, and so is an incomplete HTTP header. In the request sent by slowloris, the final [CRLF] is missing. So when the server receives this request, it starts its timer and waits until it receives two consecutive [CRLF]s or the timer expires. Thereafter, slowloris starts sending incomplete header parts just before the timer expires, and the server keeps waiting and renewing its timer each time it receives an incomplete part. If the timeout is 300s (Apache web servers default), slowloris will send the next incomplete header in 299s.

Slowloris is designed to quickly tie up a typical web server or proxy server based on a Linux Unix machine by a single attacker (since Windows limits the number of open sockets at any given time). It can lock up all its threads as the server patiently waits for more data. Slowloris can customize the timeouts simultaneously when some servers may have a smaller timeout tolerance than others. Additionally, a special function aids the attacker in finding the right-sized timeouts as well. Modern operating systems need not shut down the sockets as a slowloris attack does not consume many resources. In fact, in certain circumstances, it makes slowloris better than a typical flooder. So we can say that slowloris is the HTTP equivalent of an SYN flood.

3 Related Work

Many different approaches to counter slowloris attacks exist in the current literature. Some solutions use approaches to limit the number of connections per user or set the timeouts for each connection [16]. Apache developed a security module named mod_antiloris to protect the web server from slowloris attacks. The module limits the number of threads in the READ state on a per-IP address basis. It allows a default maximum number of 5 HTTP connections per IP address, which is modifiable by re-configuring the server. However, the server can know nothing about the attacker's IP address, and there is no basis for choosing five as the maximum limit. Sometimes, a legitimate user may need many parallel HTTP connections to download a large file quickly. He may divide the file into many parts and download them simultaneously. The requirement of the total number of connections for a user may vary and increase the maximum limit in normal conditions. Hence, limiting the maximum number of connections cannot be an appropriate solution. The IP read limit value can be changed by configuring the antiloris module. However, once set, it is fixed for that run and cannot be changed on run time. Similarly, solutions using IPTABLES [17] and advanced policy firewall (APF) [18] also limit the number of active connections. Solutions using other Apache modules, such as mod_evasive [19] and mod_qos [20] that monitor the incoming server requests and prioritize the incoming HTTP requests, also counter slowloris attacks but are not much impactful.

A combination of timeout and data rate limit per request was also used to mitigate the attack [21–23]. Tools such as Suricata [24] and SNORT [25] were also analyzed for the detection of the slowloris attack by De Sousa Araújo *et al.* [26]. The analysis reported that the number of alerts generated by Suricata to detect the attack was inadequate whereas SNORT has a trade-off between the memory and process consumption during attack detection. SNORT also reports many false positives when the number of connections per user increases based on the number of objects on a web page [27]. Schemes such as Gigabit Ethernet Secure Network Interface Controller (GESNIC) web server protection protect the server from high-performance DDoS attacks such as slowloris. It performs simple packet analysis based on the feature that a slowloris GET request sends. It drops any GET request that does not contain two CRLFs at the end of the header [28]. However, this mechanism may sometimes detect a legitimate user as an attacker if the user has a slow internet connection. In a slow internet connection, sometimes a web server may get an incomplete HTTP header, but this does not mean the sender is an attacker. Therefore, not having two CRLFs at the end of a GET request is a necessary but insufficient condition to confirm a client as an attacker. Hence, the chance of getting a false alarm is higher in GESNIC in case of a slow client connection.

Statistical and machine learning (ML) approaches were also used for the signature and anomaly-based detection of slowloris attacks [3, 29–31]. Swe *et al.* proposed a framework for the detection of slowloris attacks that uses gain-ratio and chi-squared ranking methods to select optimal feature subsets from a large dataset and then train ML algorithms [32]. Slowloris attack detection using a Deep Learning (DL) neural network in [33] is based on the statistical variations

of the traffic, that tend to generate false positives when the traffic is also from legitimate users. This happens because the slowloris attack traffic pattern is very similar to genuine traffic and discriminating between them is one of a challenge. However, Velan *et al.* pointed out that during the traffic analysis for attack detection, the process of flow data creation is generally not considered which leads to the generation of false positives [34]. Kemp *et al.* proposed a method to identify and extract features related to slowloris using Principal Component Analysis (PCA) from the Netflow data [35]. The features are then used to train several ML algorithms. However, the method was not implemented and validated in real-time. Fu *et al.* proposed a real cloud computing platform-oriented attack detection using time-frequency characteristics of traffic data [36]. They learn the potential time-frequency domain connection in the normal sequence using a deep neural network and generate the reconstructed sequence. The difference between the two sequences discriminates against the attack in a short time with high accuracy. However, the method again suffers from false alarms owing to the short duration of the connection time assumed.

Most existing related works to detect slowloris attacks are based on the feature selection and then training the ML or DL algorithms. Though the proposed works have high detection accuracy but may generate false positives when the attack traffic mimics legitimate traffic and the ML model fails to discriminate them. In this paper, we aim to detect a slowloris attack without involving any ML approach. Therefore, we propose an adaptive connection timeout approach to detect the attack and identify the attacker. Thus, we summarize the following requirement for a method to detect a slowloris attack.

(1) It must be able to both detect the attack and identify the attacker.
(2) It must be independent of the size of attack traffic and the number of attackers.
(3) The legitimate users should not be affected by any detection and mitigation mechanism adopted.

4 Motivation

The stealthy nature of the slowloris attacks and false positives in the existing solutions motivate us to contribute a solution in the current literature, which we highlight below.

(1) Existing methods for DDoS attack detection are mainly targeted on the network layer and incapable of the detection of slow, low-rate, and stealthy slowloris attacks.
(2) Most existing methods for slowloris attack detection are based on statistical features and ML that fail to discriminate between benign and legitimate traffic.
(3) Solutions that limit connections per user and timeout mechanisms impact legitimate user connections.
(4) Dependence on header format, such as a blank line at the end of a header, results in many false positives when a user has a slow connection.

5 Problem Definition and Assumptions

The slow, low-rate, and stealthy nature of the slowloris attack can easily disrupt internet services if not detected early. Also, the discrimination of the attack traffic from the legitimate traffic is a challenge since slowloris share the same characteristic of legitimate slow connection. Therefore, the main objective of this work is to develop a method that can not only detect the attack but also discriminate between the attack and normal traffic without hampering legitimate users in the process. Once the attack is detected, the method should identify the attacker's IP address for onward preventive actions. The assumptions that we consider for our detection method are as follows:

(1) Only one server in the network is the victim of the slowloris attack.
(2) All processing of the network traffic and analysis of the detection method is done on the victim-end.
(3) The attackers are distributed across multiple botnets.
(4) The attackers send incomplete/partial HTTP requests with no request body [37].

6 Our Proposed Approach

The web server timeout for completing an HTTP GET request plays a significant role in designing a near real-time detection mechanism. Adaptive timeout is an ability of a web server by which the web server can dynamically change the timeout value for an IP address in runtime. Here, we do not need to reconfigure a web server to change the timeout of any client. In our approach, we implement a mechanism in which the timeout can be changed per IP address basis in runtime. Thus, we can assign different timeout values for clients based on their behavior. Initially, all the clients will have the same timeout value, which is the default timeout for the web server.

6.1 Detection Architecture

We consider a victim-end architecture for slowloris attack detection, as shown in Figure 2. It has several advantages in attack detection. First, the par-

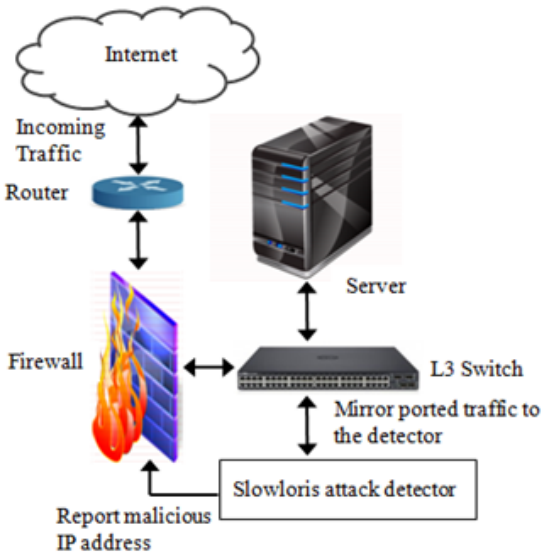

Figure 2. Detection architecture of slowloris attack

tial HTTP requests sent from the attackers appear negligible to any detection module deployed in the network edges and are very low in average volume. Thus, it is convenient to observe the traffic behavior on the victim's end. Second, it is easy to deploy. A single router connects our architecture to the Internet. All the incoming traffic reaches the server by passing through an edge router, firewall, and an L3 switch. The incoming traffic from the router/firewall is directly forwarded to the server through the switch during normal conditions. A copy of the traffic seen at one of the switch's ports is also forwarded to the Slowloris Attack Detector module through port mirroring when it forwards the traffic to the victim server. The detection module is a traffic monitoring system with high processing capacity and large memory. It passively monitors the forwarded incoming traffic to detect the slowloris attack. It reports the attacker's IP addresses back to the firewall to take preventive measures once an attack is detected.

6.2 Adaptive Timeout Method

In our approach, we divide the detection mechanism into two modules. The first module is the suspect determination module, which monitors the normal web traffic and tries to find out some potential suspects. If a suspect is found, it informs its IP address to the second module, which is the Attacker Verification module. This module verifies whether a potential suspect is an attacker or not. It uses an adaptive timeout mechanism for the suspects and records their behavior in response to the change in timeout. By monitoring the response of the suspects, it can confirm whether it is an attacker or not. Table 1 shows the symbols used in the algorithms of the modules,

Table 1. Symbols used

Symbols	Meanings
$\boldsymbol{R}$	HTTP request from a client
$\boldsymbol{addr}$	IP address of a client
A_R	Number of anomaly requests per IP address during the suspect determination
$\boldsymbol{next}$	Pointer to the next node
$\boldsymbol{L}$	List to represent the table containing anomaly requests per IP address
$\boldsymbol{H}$	The header of an IP address
σ	The threshold for the number of anomaly requests per IP address
$addr_s$	Suspect IP address
T_i	Current timeout value
$\boldsymbol{C}$	Maximum number of half-open requests the server can store
δ_r	The reduction rate of timeout
A_i	Total number of anomaly requests per IP address per timeout iteration
A_T	Total number of anomaly requests per IP address during suspect verification

and Figure 3 depicts the flowchart for the proposed detection method.

6.2.1 Suspect Determination Module

This module maintains an IP Specific Anomaly Request (IPSAR) table, which contains two columns: IP address and integer number. The IP address represents the address of a web client, and the integer number represents the corresponding total number of malicious requests sent by that client. The total number of malicious requests is calculated based on the most recent hundred requests made by that client. All the HTTP headers that do not contain two CRLFs are taken as malicious requests and the corresponding number A_R for that IP address is incremented in the table. We call this table the First Anomaly table. The anomaly calculation function continuously updates the IPSAR table for all the requests coming to the web server. The monitoring function in the module constantly monitors all the entries of the IPSAR table. A threshold value σ is set based on the various laboratory experiments. If the number of anomaly requests for any IP address is greater than the threshold σ, then the monitoring function signals the verification module with the IP address to verify whether it is an attack or not. The pseudocode of the suspect determination module is presented in Algorithm 1.

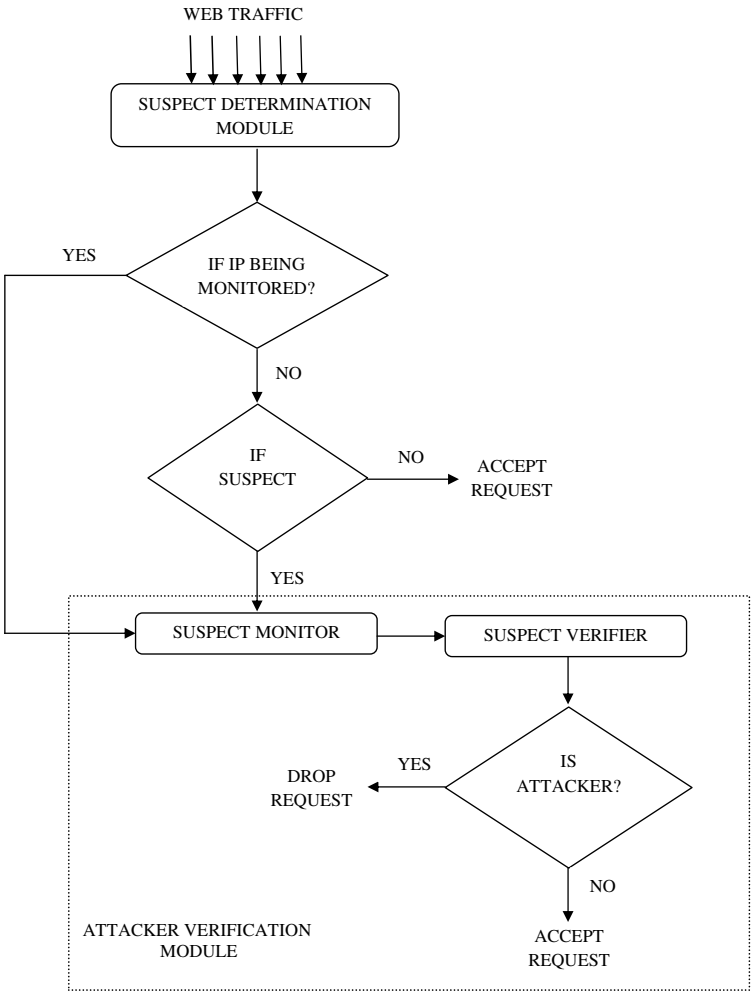

Figure 3. Flowchart for the proposed detection method

6.2.2 Attacker Verification Module

This module uses an adaptive timeout approach to confirm whether a suspect IP address is an attacker. A normal small-scale web server has the capacity of serving 1000 read requests simultaneously. Therefore, our approach must confirm the attacker's IP address as an attacker before it consumes all the server resources. We give our verification module enough time to collect adequate evidence to confirm an attacker with a significantly low false positive rate (FPR). The module repeatedly reduces the timeout value for a suspect IP address and is simultaneously updated in the server. Since the incoming traffic is mirror-ported to both the server and the detection module through the switch, the behavior or response of the suspect IP address for each reduced timeout is recorded. The response is recorded in terms of the total number of anomaly requests corresponding to each timeout value. The module is based on the idea that an attacker may open multiple connections and send incomplete/partial HTTP requests to respond to the reduced timeout values for each connection. Therefore, the module is divided into two parts: (i) Suspect Monitor and (ii) Suspect Verifier or Confirmation module, which we describe in the next subsections.

6.2.2.1 Suspect Monitor

In this part, the timeout value for the suspect is reduced by using an adaptive timeout mechanism. Figure 4 depicts the flowchart for the suspect monitor of a potential attacker. It consists of a recursive function that iteratively reduces the timeout value by δ_r percent for the suspected IP address $addr_s$. For each reduced timeout, it waits for $2*\delta_r$ number requests to come from the $addr_s$, which is a response of the $addr_s$. If $2*\delta_r$ number of requests arrives from the $addr_s$, it indicates an attacker else, a slow user. This module stores the timeout value T_i and the corresponding number of anomaly requests A_i for all the iterations in a table. The iterations are stopped when the total anomaly request A_T crosses half of the capacity C of the web server i.e., $A_T > C/2$. Thereafter, it in-

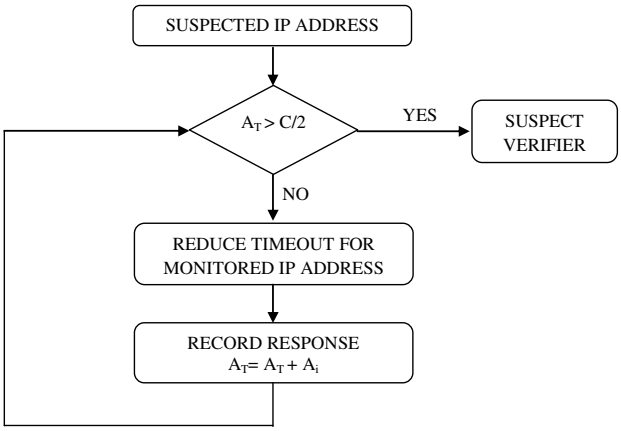

Figure 4. Flowchart of suspect monitor

Algorithm 1 Suspect Determination

Input: R
Output: Suspected IP address

1: Define a linked list $IPSAR$ with the following attributes: *struct IPSAR {Address addr, int* A_R*, struct IPSAR *next}*
2: Create a list L to represent the table containing Anomaly Request per IP address: *struct IPSAR* $*L$
3: Extract Header H from Request R
4: Extract IP address *addr* of the Sender from H
5: **if** *addr* already exists in the list **then**
6: Goto Step 6
7: **else**
8: Create an entry for *addr* in the list
9: **end if**
10: Check H for two $CRLFs$
11: **if** H contains to two $CRLF$ **then**
12: Request R is Accepted
13: **else**
14: $A_R = A_R + 1$
15: **end if**
16: **if** $A_R > \sigma$ **then**
17: Send the IP address to the verification module
18: **end if**
19: Repeat Steps 3 to 16 for all the HTTP Requests coming to the server

vokes the suspect verifier module. The pseudocode of the suspect monitoring and verification algorithm is presented in Algorithm 2.

6.2.2.2 Suspect Verifier

In this part, the module checks the response of the $addr_s$ based on the different reduced timeout values T_i and the anomaly requests values A_i. Figure 5 shows the reduced timeout for different δ_r along with the expected number of packets and the received packets A_i from $addr_s$. If the module observes constant attempts A_i from the $addr_s$ to send a packet for every reduced timeout T_i, it then confirms that $addr_s$ as an attacker. Otherwise, it considers a legitimate user whose Internet connection is very slow. This is because an attacker will always attempt to send incomplete/partial HTTP requests for each connection opened by it in response to the reduced timeout. Whereas, a legitimate user with a slow Internet connection will not change its behavior in response to such a reduction. Due to the reduced timeout, such slow users may open new connections, but their behavior will remain unchanged and continue to timeout. Unlike this, an attacker will attempt to evade timeout by sending partial requests just before the timeout for its connections and keep them open for as long time as possible. Though initially, the slow users may be marked as suspicious by the detection module. But eventually, due to their unchanged behavior, they are marked as legitimate and are not

Algorithm 2 Adaptive Timeout-based Suspect Monitoring and Verification

Input: $addr_s, T_i, C, L$
Output: Anomaly array and Timeout array

1: Declare A_T
2: $A_T = L \rightarrow A_R$
3: **if** $A_T > C/2$ **then**
4: Goto Suspect Verifier/Confirmation module
5: **end if**
6: $T_i = T_i - (T_i * \delta_r)$
7: Wait for $2 * \delta_r$ number of requests to come from $addr_s$
8: Calculate the total number of anomaly requests A_i during this period and store it in the anomaly array
9: Store the timeout value T_i in the timeout array
10: $A_T = A_T + A_i$
11: Goto Step 3

δ_r=20%		
Timeout	**Expected packet**	**A_i**
60	0.4	1
48	0.4	1
38	0.4	1
31	0.4	1
25	0.4	1
20	0.4	1
16	0.4	1
13	0.4	1
10	0.4	1
8	0.4	1
6	0.4	1
5	0.4	1
4	0.4	1
3	0.4	1
3	0.4	1
2	0.4	1
1	0.4	1

δ_r=40%		
Timeout	**Expected packet**	**A_i**
60	0.8	1
36	0.8	1
22	0.8	1
13	0.8	1
8	0.8	1
5	0.8	1
3	0.8	1
2	0.8	1
1	0.8	1

δ_r=80%		
Timeout	**Expected packet**	**A_i**
60	1.6	2
12	1.6	2
2	1.6	2

Figure 5. Expected and anomaly packets A_i with respect to the reduced timeouts δ_r

reported to the firewall to block their IP addresses.

7 Experimental Results

For testing our proposed algorithm, we use our Tezpur University Web Server dataset with the legitimate web traffic of various users. Each tuple of the dataset contains seven attributes, viz. the IP address of the user, date, time, the time difference with GMT, main message, body, HTTP protocol version, and server response code. One example tuple from the dataset is shown below.

66.54.88.131 –[26/Oct/2022:04:10:56 + 0530] "GET /hostels/swh/prevAP.html HTTP/1.1" 200 7042

In the above example, *66.54.88.131* is the IP address of the client sending the request, *26/Oct/2022* is the date, *04:10:56* is the time of arrival of the request, *+0530* is the time difference in GMT for India, *GET /hostels/swh/prevAP.html* is the main request, *HTTP/1.1* is the protocol version, and *200* is the server response code. For our algorithm, we need only a few attributes, such as the IP address of the user, time of arrival of the request, main message body, and HTTP protocol version. Therefore, we extract the required attributes from the dataset by cleaning the raw data by writing C programs. One example tuple from the dataset is shown below.

66.54.88.131 04:10:56 GET /hostels/swh/prevAP.html HTTP/1.1

To process this data properly in our program, we add another tuple to represent CR and LF. An example tuple from the final set is:

66.54.88.131 04:10:56 GET /hostels/swh/prevAP.html HTTP/1.1 CRLF CRLF

We then generate the attack traffic in our laboratory programmatically. To generate the attack requests, we use the standard slowloris tool [38]. We simulate the different attack behaviors by writing various attacker programs. One example tuple from the dataset is given below.

127.0.0.2 04:10:50 GET /server/img01.jpg CRLF

In our experiment, we consider three attackers and five normal users and generate the traffic for different scenarios. After generating the attack traffic, we combine the attack traffic with the normal web traffic captured from our University web server. The normal web traffic obtained from the server is processed and restricted to only five normal users. We generate different datasets for different scenarios, and the number of attacks and normal request traffic in the datasets varies between 20000 and 300000. This number depends on the number of normal users' slow requests and partial HTTP requests sent by the attackers for various scenarios. We sample the dataset into 10s consecutive time intervals where some sam-

Table 2. FPR for different threshold values (σ)

Threshold Value σ	FPR (in %)	Specificity (TNR in %)
100	3.33	96.67
150	3.35	96.65
200	3.23	96.77
250	3.19	96.81
300	3.13	96.87
350	3.01	96.99
400	2.97	97.03
450	2.94	97.06
500	2.83	97.17
550	2.71	97.29
600	2.79	97.21
650	2.61	97.39
700	2.57	97.43
750	2.41	97.59
800	2.63	97.37
850	2.74	97.26
900	2.81	97.19
950	3.06	96.94
1000	3.27	96.73

ples may not contain the malicious requests based on the scenarios. Each 10s sample consists of 500-1000 average request packets, and the total test duration ranges between 30-50 mins for different scenarios. In a slowloris attack, the traffic rate is very low, and an attacker sends a partial HTTP request just before the timeout. We consider a default timeout of 60s for the Apache 2.4 web server. Therefore, in the datasets, the attackers send partial HTTP requests depending on the reduced timeout to keep the connections open for a long time. The time interval for sending such follow-up partial requests is not fix. Thus, the presence of such malicious requests is random across the samples. We process our algorithm in the above datasets by writing various C and Java programs. The algorithm keeps track of the number of anomaly requests A_R per IP address to find the suspicious IP address, and thereafter, the subsequent modules confirm the IP address as an attacker or legitimate. We test our algorithm for the different scenarios and observe that it can detect the attackers properly with a much lower FPR. We record the change in FPR and TNR for different threshold values σ as shown in Table 2 and Figure 6.

From the above graph, we see that if we increase the threshold value, then the FPR decreases. However, if

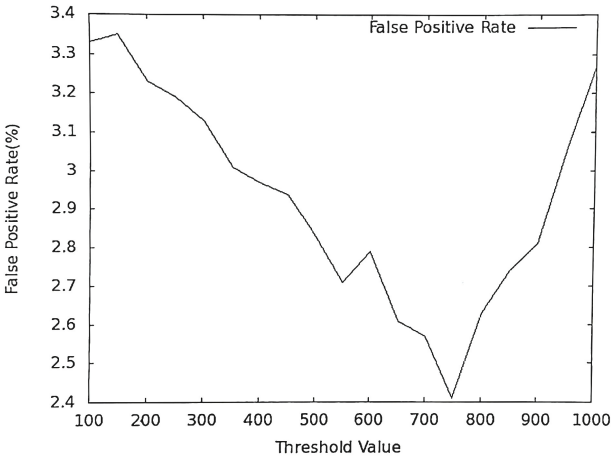

Figure 6. FPR for different threshold values σ

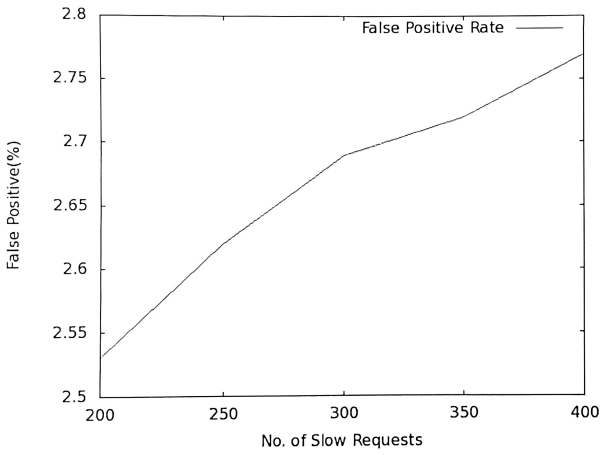

Figure 7. Change in FPR for different numbers of slow requests

we increase further, FPR tends to increase. Therefore, we can conclude that the threshold value can neither be very small nor it can be very large. From the above graph, we can compute the threshold value of our web server. We design various test cases to observe the change in FPR for five slow users who are not attackers, as shown in Table 3 and Figure 7.

Table 4 and Table 5 show the change in FPR for the number of slow users. In these scenarios, we fix the total number of malicious requests for these slow users. The same is depicted in Figure 8 and Figure 9.

From the above results, we observe that the FPR increases with the increase in the number of slow requests. That means, if some of the valid users are very slow, then our method may also detect them as an attacker. Table 6, Table 7, and Table 8 show the detection accuracies of our method for various test cases or scenarios that we have run in our laboratory. We increased the number of malicious requests from

Table 3. FPR for different numbers of slow requests

Normal user	Number of attacker requests	Numbers of slow requests	FPR (in %)	Specificity (TNR in %)	Sensitivity (TPR in %)
User 1	1000	200	2.53	97.50	97.40
User 2	1000	250	2.62	97.20	97.30
User 3	1000	300	2.69	97.33	97.30
User 4	1000	350	2.72	97.14	97.20
User 5	1000	400	2.77	97.25	97.30

Table 4. FPR for different numbers of slow requests

Normal user	Number of attacker requests	Numbers of slow requests	FPR (in %)	Specificity (TNR in %)	Sensitivity (TPR in %)
User 1	5000	500	2.38	97.60	97.62
User 2	5000	600	2.50	97.50	97.50
User 3	5000	700	2.68	97.32	97.32
User 4	5000	800	2.57	97.43	97.43
User 5	5000	900	2.79	97.21	97.21

Table 5. FPR for different numbers of slow requests

Normal user	Number of attacker requests	Numbers of slow requests	FPR (in %)	Specificity (TNR in %)	Sensitivity (TPR in %)
User 1	10000	600	2.41	97.50	97.59
User 2	10000	800	2.55	97.50	97.45
User 3	10000	1000	2.47	97.50	97.53
User 4	10000	1200	2.73	97.25	97.27
User 5	10000	1400	2.81	97.21	97.19

Table 6. Detection accuracies for Test case I

Total number of requests	Number of malicious requests	Numbers of slow requests	Number of Suspect IP address	Number of detection	FPR (in %)	Detection accuracy
25933	1000	200	1200	1026	2.53	97.47
259383	1000	250	1250	1027	2.63	97.37
259433	1000	300	1300	1033	2.69	97.31
259483	1000	350	1350	1039	2.72	97.28
259533	1000	400	1400	1035	2.70	97.30

Table 7. Detection accuracies for Test case II

Total number of requests	Number of malicious requests	Numbers of slow requests	Number of Suspect IP address	Number of detection	FPR (in %)	Detection accuracy
263633	5000	500	5500	5122	2.38	97.62
263733	5000	600	5600	5129	2.50	97.50
263833	5000	700	5700	5138	2.68	97.32
263933	5000	800	5800	5132	2.57	97.43
264033	5000	900	5900	5144	2.79	97.21

Table 8. Detection accuracies for Test case III

Total number of requests	Number of malicious requests	Numbers of slow requests	Number of Suspect IP address	Number of detection	FPR (in %)	Detection accuracy
268733	10000	600	10600	10247	2.41	97.59
268933	10000	800	10800	10262	2.55	97.45
269133	10000	1000	11000	10254	2.47	97.53
269333	10000	1200	11200	10281	2.73	97.27
269533	10000	1400	11400	10293	2.81	97.19

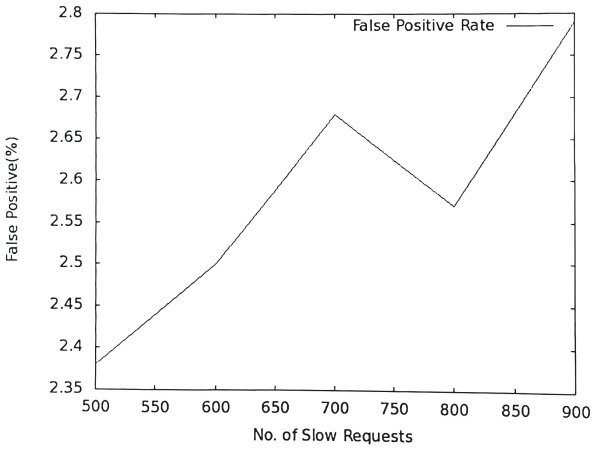

Figure 8. Change in FPR for different numbers of slow requests

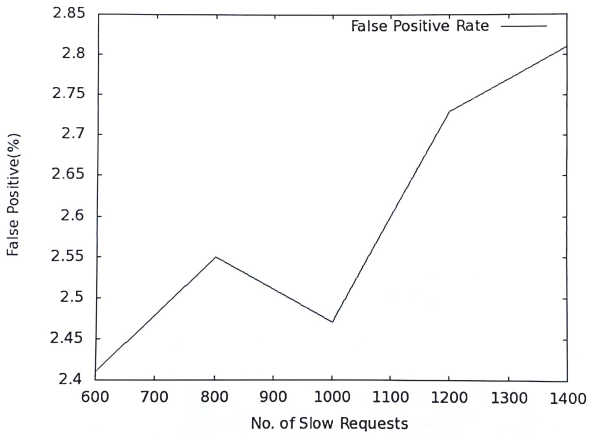

Figure 9. Change in FPR for different numbers of slow requests

the three attackers as well as the number of legitimate slow requests from five users. We observe that our algorithm gives very small false positive results. The detection accuracies remain almost consistent throughout the different test cases when we increase the number of slow requests and the attackers are detected in a few seconds. However, in some scenarios, the FPR increases when we increase the number of slow user requests, which is negligible.

8 Conclusion and future work

Protocol-specific vulnerability attack is a new trend of attack in the application layer. The works done in the research literature address mainly flooding-based attacks such as Net-DDoS and App-DDoS attacks. These methods are incapable of detecting sophisticated low and slow protocol-specific vulnerability attacks. Solutions related to protocol exploitation attacks such as slowloris are not many. So, there is a need for a detection system capable of near real-time detection of these attacks with a very low FPR. Some of the existing detection mechanisms emphasize attack detection by using unsupervised anomaly detection techniques, but they do not consider some other important issues like real-time detection or false alarms. In this work, we have tried to detect slowloris attacks in near real-time with a very low false alarm rate and high detection accuracy by using an anomaly-based suspect determination and an adaptive timeout-based detection mechanism. The proposed method cannot not only detect the attack but also identify the attacker's IP address and is independent of the size of the attack and the attacker numbers. Also, the proposed method can discriminate between the legitimate and the attacker's traffic with very low FPR, thereby not impacting the legitimate users. Significant recent trends include vulnerability attacks of various famous application layer attacks. Existing works have given various approaches to tackle those attacks individually or as a group. However, we need one single system that can protect a server from most of the application layer attacks. Our future research aims to build a single defense system that can detect and mitigate different types of App-DDoS attacks.

Acknowledgment

All the studies and experiments were carried out in the Laboratory of the Department of Computer Science and Engineering, Tezpur University, Napaam, Tezpur, Assam, India. This study did not receive funding from any agencies. The authors declare that there is no conflict of interest regarding the publication of this paper.

References

[1] S Prabha and R Anitha. Mitigation of application traffic ddos attacks with trust and am based hmm models. *International Journal of Computer Applications*, 6:26–34, 2010.

[2] P J Criscuolo. Distributed denial of service, tribe flood network 2000, and stacheldraht. *CIAC-2319, Department of Energy Computer Incident Advisory Capability (CIAC), UCRL-ID-136939, Rev. 1., Lawrence Livermore National Laboratory*, 1, 2000.

[3] V Durcekova, L Schwartz, and N Shahmehri. Sophisticated denial of service attacks aimed at application layer. In *ELEKTRO*, pages 55–60. IEEE, 2012.

[4] X Xu, X Guo, and S Zhu. A queuing analysis for low-rate dos attacks against application servers. In *IEEE International Conference on Wireless Communications, Networking and Information Security*, pages 500–504. IEEE, 2010.

[5] J Mirkovic and P Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communications Review*, 34:39–53, 2004.

[6] C Douligeris and A Mitrokotsa. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44:643–666, 2004.

[7] T Peng, C Leckie, and K Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Survey*, 39:3–es, 2007.

[8] RioRey. Taxonomy of ddos attacks. 2022.

[9] S Ranjan, R Swaminathan, M Uysal, and E Knightly. Ddos-resilient scheduling to counter application layer attacks under imperfect detection. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–13. IEEE, 2006.

[10] Wallarm. What is slowloris attack. 2022.

[11] R Barnett. (updated) modsecurity advanced topic of the week: Mitigating slow http dos attacks. *TrustWave*, 2011.

[12] K J Higgins. Researchers to demonstrate new attack that exploits http. *DarkReading*, 2010.

[13] S Shekyan. Are you ready for slow reading? *Qualys*, 2012.

[14] J Pelline. Mydoom downs sco site. *CNET*, 2004.

[15] S Pillai. Slowloris http dos attack and prevention. */ROOT.IN*, 2013.

[16] R Papadie and I Apostol. Analyzing websites protection mechanisms against ddos attacks. In *9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6. IEEE, 2007.

[17] R K Sharma, B Issac, and H K Kalita. Intrusion detection and response system inspired by the defense mechanism of plants. *IEEE Access*, 7:52427–52439, 2019.

[18] V Jyothi, X Wang, S K Addepalli, and R Karri. Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks. In *29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pages 587–588. IEEE, 2016.

[19] S Sivabalan and P J Radcliffe. Feasibility of eliminating idps devices from a web server farm. *International Journal of Network Security*, 20:433–438, 2018.

[20] R Giunta, F Messina, G Pappalardo, and E Tramontana. Augmenting a web server with qos by means of an aspect-oriented architecture. In *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 179–184. IEEE, 2012.

[21] T Shorey, D Subbaiah, A Goyal, A Sakxena, and A K Mishra. Performance comparison and analysis of slowloris, goldeneye and xerxes ddos attack tools. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 318–322. IEEE, 2018.

[22] E Damon, J Dale, E Laron, J Mache, N Land, and R Weiss. Hands-on denial of service lab exercises using slowloris and rudy. In *Proceedings of the 2012 Information Security Curriculum Development Conference*, pages 21–29. ACM, 2012.

[23] N Sultana, S Bose, and B T Loo. An extensible evaluation system for dos research. In *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*, pages 344–351. IEEE, 2019.

[24] W Park and S Ahn. Performance comparison and detection analysis in snort and suricata environment. *Wireless Personal Communication*, 94:241—-252, 2017.

[25] D J Day and B M Burns. A performance analysis of snort and suricata network intrusion detection and prevention engines. In *Proceedings of the Fifth International Conference on Digital Society*, pages 187—-192. IARIA, 2011.

[26] T E de Sousa Araújo, F M Matos, and J A Moreira. Intrusion detection systems' performance for distributed denial-of-service attack. In *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 1–6. IEEE, 2017.

[27] V da Silva Faria, J A Gonçalves, C A M da Silva, G de Brito Vieira, and D M Mascarenhas. Sdtow: A slowloris detecting tool for wmns. *Information*,

11:544, 2020.

[28] H Kim, B Kim, D Kim, I K Kim, and T M Chung. Implementation of gesnic for web server protection against http get flooding attacks. In *International Workshop on Information Security Applications. WISA 2012. Lecture Notes in Computer Science*, pages 285–295. Springer, 2012.

[29] M Sikora, T Gerlich, and L Malina. On detection and mitigation of slow rate denial of service attacks. In *Proceedings of the 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 1–5. IEEE, 2019.

[30] J Kim and H S Kim. Intrusion detection based on spatiotemporal characterization of cyberattacks. *Electronics*, 9:460, 2020.

[31] K J Singh and T De. Mlp-ga based algorithm to detect application layer ddos attack. *Journal of Information Security Applications*, 36:145–153, 2017.

[32] Y M Swe, P P Aung, and A S Hlaing. A slow ddos attack detection mechanism using feature weighing and ranking. In *Proceedings of the 11th Annual International Conference on Industrial Engineering and Operations Management*, pages 4500–4509. IEOM Society International, 2021.

[33] A Al-Harbi and R Jabeur. An efficient method for detection of ddos attacks on the web using deep learning algorithms. *International Journal of Advanced Trends in Computer Science and Engineering*, 10:2821–2829, 2021.

[34] P Velan and T Jirsik. On the impact of flow monitoring configuration. In *Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE, 2020.

[35] C Kemp, C Calvert, T M Khoshgoftaar, and J L Leevy. An approach to application-layer dos detection. *Journal of Big Data*, 10:1–30, 2023.

[36] Y Fu, X Duan, K Wang, and B Li. Low-rate denial of service attack detection method based on time-frequency characteristics. *Journal of Cloud Computing: Advances, Systems and Applications*, 11:1–19, 2022.

[37] V Sundar. What is slowloris ddos attack and how does it work? *Indusface*, 2023.

[38] G Yaltirakli. Slowloris 0.2.6. low bandwidth dos tool. *Github*, 2023.

Dr. Kangkan Talukdar is working as an Officer cum Network Administrator in Numaligarh Refinery Limited. He received his M.S. degree from Tezpur University, Assam, India, in 2014 and his Ph.D. degree from the Indian Institute of Technology, Guwahati, in 2021. His research interests include Network Security, ML, Deep Learning, and Internet of Things (IoT).

Dr. Debojit Boro is working as an Associate Professor in the Department of Computer Science and Engineering at Tezpur University, Assam, India. He received his Ph.D. degree from Tezpur University Assam, India, in 2017. His research interest includes Network Security, Ensemble Classifiers, Evolutionary Computation, ML, Internet of Things (IoT), Tactile Internet, and Blockchain. He is a reviewer of the EURASIP Journal of Information Security and has reviewed several of its papers.