Journal of Information System and Application Development
Volume 3 Number 1 Maret 2025 page 58-67
P-ISSN: 2988-5698 | E-ISSN: 2988-4721
https://jurnal.unmer.ac.id/index.php/jisad

Implementation of rate limiting and Telegram bot for HTTP GET Flood attack mitigation

Bagas Satya Dharma*, Ahmad Rofiqul Muslikh
Program Studi Sistem Informasi, Fakultas Teknologi Informasi, Universitas Merdeka Malang,
Jl. Terusan Dieng No. 62-64, Malang, 65146, Indonesia

E-mail: *bagassatya2002@gmail.com

Abstract. Distributed Denial of Service (DDoS) attacks pose a serious cybersecurity threat by overwhelming web servers with excessive traffic, rendering them inaccessible. One of the most common types is HTTP Flood, where massive HTTP GET and POST requests continuously drain server resources, leading to performance degradation or system failure. This study aims to analyze the impact of HTTP Flood DDoS attacks on web servers and evaluate the effectiveness of mitigation strategies using firewalls, rate limiting, and Telegram bot notifications. The research was conducted through experimental testing on an Apache server hosted on a Digital Ocean VPS, where server performance was measured before and after mitigation. The results indicate that a combination of firewalls configured with iptables and rate limiting successfully reduced CPU load by over 90%, maintaining server stability even under attack. Additionally, Telegram bot played a crucial role in real-time attack detection and response, enabling administrators to take immediate action. In conclusion, the applied mitigation techniques effectively reduced the impact of DDoS attacks and enhanced server resilience.

Keywords: cyber security, denial of service, rate limiting

Submitted: 11-03-2025 | **Accepted:** 06-04-2025 | **Published:** 21-04-2025

How to Cite:
B. S. Dharma and A. R. Muslikh, "Implementation of rate limiting and Telegram bot for HTTP GET Flood attack mitigation," *Journal of Information System and Application Development (JISAD)*, vol. 3, no. 1, pp. 58-67, 2025, doi: 10.26905/jisad.v3i1.15398.

INTRODUCTION

In the digital era, information and communication technology plays a vital role in various aspects of life. One of the most prominent forms of its utilization is through the web, which has become a primary infrastructure for providing information, public services, and online transactions. The web refers to a collection of interconnected digital pages within a domain, presenting various types of data such as text, images, audio, video, and animations. All of this information can be accessed globally via the internet [1].

As dependence on the web increases, so do the challenges related to its availability and security. One of the main threats that can disrupt services is a Distributed Denial of Service (DDoS) attack [2]. A DDoS attack is a type of cyberattack that floods a server with requests from multiple devices simultaneously [3]. Unlike conventional Denial of Service (DoS) attacks, DDoS attacks are distributed and can lead to performance degradation or even complete service outages [4], [5].

One common form of DDoS attack is the HTTP Flood, which targets the application layer [6]. HTTP Flood DDoS attacks use seemingly normal HTTP GET or POST requests to overwhelm the server, making them difficult to detect as malicious activity. These attacks can cause the server to go down [7], [8].

Various studies have been conducted to develop mitigation methods against DDoS attacks. Research [9] shows that proper firewall configuration can significantly reduce the impact of DDoS attacks by lowering server CPU load from nearly 100% to 25.3%. Another widely used mitigation strategy is rate limiting, a mechanism that regulates the maximum number of requests allowed from a single IP address within a certain time frame [10]. According to [11], the implementation of an adaptive weight-based rate limiting mechanism in a Software Defined Network (SDN) architecture, integrated with Graph Neural Networks (GNN) modeling, enables dynamic monitoring of host activity rates and penalizes anomalous behavior, effectively reducing the impact of DDoS attacks on the performance of network controllers and switches.

In addition to firewall- and rate limiting-based mitigation, a major challenge in handling DDoS attacks is rapid detection and response. Administrators often become aware of an attack only after the server's performance degrades or goes down, leading to delayed responses. Therefore, a real-time threat detection system that can automatically notify administrators is needed to enable immediate action. To address this challenge, this research proposed using a Telegram bot as an automatic notification system in DDoS attack mitigation [12]. By integrating firewall, rate limiting [13], and a Telegram-based notification system, this study aims to analyze the impact of DDoS attacks on web server performance and evaluate the effectiveness of the combined mitigation techniques.

METHOD

This study employs an experimental method, which focuses primarily on testing. The experimental method involves conducting controlled trials to observe the effects of specific variables on system performance. This research includes simulation of HTTP GET Flood attacks and measurement of the impact after mitigation techniques using rate limiting and Telegram bot notifications. The stages of the research can be seen in Figure 1.

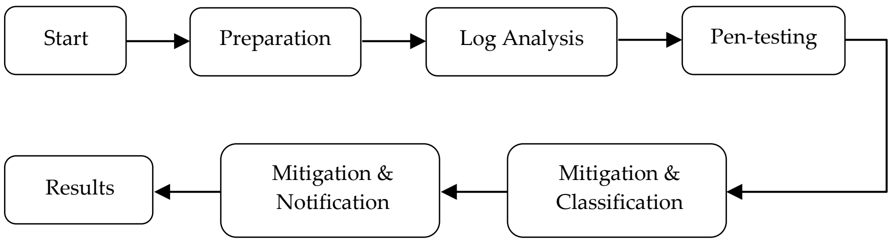


Figure 1. The Research Stages

Preparation

Web preparation is done to ensure that all testing activities take place within legal limits without involving unlawful activities. The preparation includes utilized Digital Ocean VPS as a web server by creating Ubuntu 24.10 x64-based droplets, as well as using the SSH protocol for secure remote access and management of the server. Apache Web Server [14] is also installed and run on a Digital Ocean VPS using an Apache Web Server. This allows to manage the activation and deactivation of various additional features as needed.

Log Analysis

Log analysis is used to evaluate the effectiveness of mitigation techniques on web server, by monitoring all types of requests, including normal traffic and DDoS attacks. The analysis process is carried out in three stages: before the attack, during penetration testing (pen-testing), and after mitigation is applied, to assess the server's response and the success of the mitigation. In this study, the tools used to conduct log analysis are as follows [15].

1. *tail -f /var/log/apache2/access.log*
   - ✓ *tail*: displays the last line of a file
   - ✓ -f (*follow*): monitors files continuously and displays new lines when files are updated
   - ✓ */var/log/apache2/access.log:* a log file that keeps a record of all requests received by the Apache server.

2. HTOP
   The htop command is an interactive system monitoring tool in Linux that provides real-time information related to CPU usage, memory, running processes, and other system resources [16].
3. *cat /var/log/apache2/access.log | toilet -l*
   This command is used to calculate the number of rows in the log file *access.log* belonging to the *Apache* server. Each row in this file represents one request received by the server
   - ✓ *cat /var/log/apache2/access.log*: displays the entire contents of the *access.log* file containing all requests to the Apache server
   - ✓ | *(pipe):* switches (pipes) the output from the first command to the next
   - ✓ WC -L: stands for Word Count

Pen-testing

The test of attacks that will be carried out in this study uses a script that is deliberately designed to carry out attacks. The goal is to test the resilience and response of the web server to controlled attack scenarios. This technique allows to evaluate the effectiveness of the mitigation system that has been implemented.

Mitigation and Classification

The mitigation that will be implemented uses a firewall [17] to filter the requests received by the web server. The goal is to distinguish between normal requests and those indicated as attacks. A script to set boundaries is also configured with a classification method. Next, the classification aims to compile the rules applied to the firewall. It works by separating each request received by a web server whether it is a legitimate request from a user or a request from an attack.

1. Normal user requests:
   - Regular websites: 1-10 requests per minute; Ordinary users open multiple pages or load images.
   - Interactive application: 10-30 requests per minute; Users are actively exploring various features of the app.
   - APIs and mobile apps: 0-60 requests per minute; users generate requests for data updates in real-time.
2. Requests with potential DDoS attacks:
   - Volumetric attacks: thousands to tens of thousands of requests per second from one or more IP addresses, aiming to overwhelm the server.
   - High requests from a single IP: More than 100 requests per second from a single IP address consistently, which could indicate an attack.
   - Error flood: a large spike in error status codes (such as 404 or 500) that coincide with a spike in requests signals a potential attack.

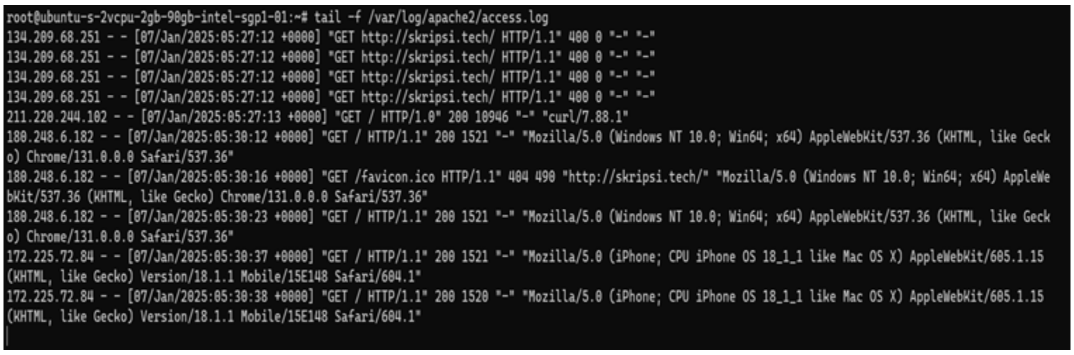
Mitigation and Notification

In this study, the mitigation script was equipped with the ability to send notifications to Telegram bots. By utilizing the Telegram API, this research integrated the bot so that whenever a web server is attacked, a notification will be received directly through the bot.

RESULTS AND DISCUSSION

Preliminary Stage

In the initial analysis stage, the researcher used a log reader tool that could provide information. The initial purpose of this log analysis is to understand the condition of the server thoroughly before any additional configurations or potential attacks. Figure 2 shows the results of the *tail command -f /var/log/apache2/access.log*. This command allows real-time monitoring of incoming requests from various IP addresses to the Apache server.



```
root@ubuntu-s-2vcpu-2gb-90gb-intel-sgp1-01:~# tail -f /var/log/apache2/access.log
134.209.68.251 - - [07/Jan/2025:05:27:12 +0000] "GET http://skripsi.tech/ HTTP/1.1" 400 0 "-" "-"
134.209.68.251 - - [07/Jan/2025:05:27:12 +0000] "GET http://skripsi.tech/ HTTP/1.1" 400 0 "-" "-"
134.209.68.251 - - [07/Jan/2025:05:27:12 +0000] "GET http://skripsi.tech/ HTTP/1.1" 400 0 "-" "-"
134.209.68.251 - - [07/Jan/2025:05:27:12 +0000] "GET http://skripsi.tech/ HTTP/1.1" 400 0 "-" "-"
211.220.244.102 - - [07/Jan/2025:05:27:13 +0000] "GET / HTTP/1.0" 200 10946 "-" "curl/7.88.1"
180.248.6.182 - - [07/Jan/2025:05:30:12 +0000] "GET / HTTP/1.1" 200 1521 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/131.0.0.0 Safari/537.36"
180.248.6.182 - - [07/Jan/2025:05:30:16 +0000] "GET /favicon.ico HTTP/1.1" 404 490 "http://skripsi.tech/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
bKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36"
180.248.6.182 - - [07/Jan/2025:05:30:23 +0000] "GET / HTTP/1.1" 200 1521 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/131.0.0.0 Safari/537.36"
172.225.72.84 - - [07/Jan/2025:05:30:37 +0000] "GET / HTTP/1.1" 200 1521 "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 18_1_1 like Mac OS X) AppleWebKit/605.1.15
(KHTML, like Gecko) Version/18.1.1 Mobile/15E148 Safari/604.1"
172.225.72.84 - - [07/Jan/2025:05:30:38 +0000] "GET / HTTP/1.1" 200 1520 "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 18_1_1 like Mac OS X) AppleWebKit/605.1.15
(KHTML, like Gecko) Version/18.1.1 Mobile/15E148 Safari/604.1"
```

Figure 2. Request Logs Before Pen-testing and Mitigation

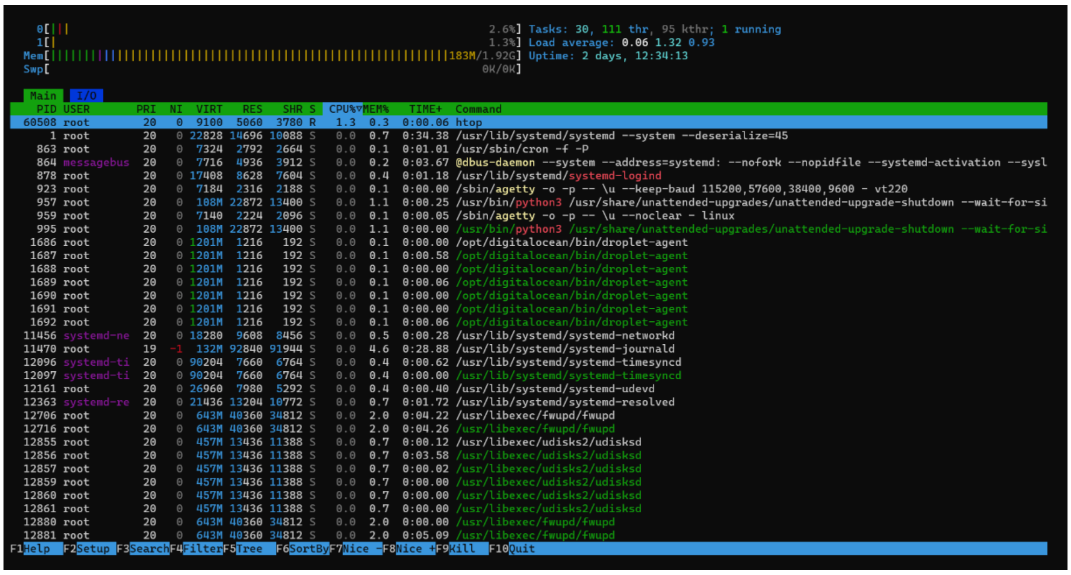
Requests from multiple IPs can be seen in Table 1. Table 1 shows script 2 and script 4 successfully accessing the main page (status 200). Meanwhile, script 1 and script 3 fail because the resource is not found (404 status), i.e. the main page and favicon. Requests for such scripts come from different types of user agents, such as the curl auto tool, the Google Chrome browser on Windows, and Safari on iPhone.

Table 1. IP Requests on the Server

No.	Script	Information
1.	*134.209.68.251 - - [07/Jan/2025:05:27:12 +0000] "GET http://skripsi.tech/ HTTP/1.1" 404 0 "-" "-"*	**IP Address:** 134.209.68.251 (visitor's address) **Date & Time:** 07/Jan/2025:05:27:12 +0000 **HTTP method:** GET (page request) **URL:** http://skripsi.tech/ **HTTP version:** HTTP/1.1 **Code Status:** 404 → Page not found **Response Size:** 0 bytes **User-Agent:** "-" → No browser information
2.	*201.220.204.201 - - [07/Jan/2025:05:27:13 +0000] "GET/HTTP/1.1" 200 10946 "-" "curl/7.88.1"*	**IP Address:** 201.220.204.201 **Request:** GET / → Accessing the main page **Code Status:** 200 → Successful **Response Size:** 10946 bytes **User-Agent:** "curl/7.88.1" → Using the curl tool to access the server
3.	*180.148.0.182 - - [07/Jan/2025:05:36:16 +0000] "GET/favicon.ico HTTP/1.1" 404 490 "http://skripsi.tech/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36"*	**IP Address:** 180.148.0.182 **Request:** GET /favicon.ico → Request site icon (favicon) **Code Status:** 404 → Favicon not found **User-Agent:** Google Chrome Browser on Windows 10
4.	*172.225.72.81 - - [07/Jan/2025:05:36:37 +0000] "GET/HTTP/1.1" 200 1521 "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 18_1_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.1.1 Mobile/15E148 Safari/604.1"*	**IP Address:** 172.225.72.81 **Request:** GET / → Accessing the main page **Code Status:** 200 → Successful **User-Agent:** Safari Browser on iPhone (iOS 18.1.1)

Figure 3 shows the results of the HTOP command before pen-testing. The system condition shown in Figure 3 reflects a relatively low resource usage and stable performance. The CPU usage is minimal at only 2.6%, indicating the server is not under heavy computational load. Memory usage is approximately 18.6%, with 183 MB used out of 981 MB available, while no swap memory is currently being utilized (0B/0B), suggesting efficient memory management. There are 30 active processes, 111 threads, 95 idle processes, and 1 process currently running, indicating moderate activity on the system.

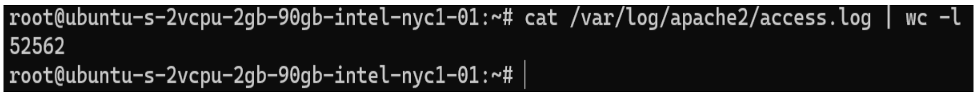
The average CPU load over the past 1, 5, and 15 minutes is 0.06, 1.32, and 0.93 respectively, which points to a generally light workload with a brief period of higher activity. Additionally, the server has been operational for 2 days, 12 hours, 34 minutes, and 13 seconds, demonstrating consistent uptime.



```
    0[|||                                                                2.6%] Tasks: 30, 111 thr, 95 kthr; 1 running
    1[|                                                                  1.3%] Load average: 0.06 1.32 0.93
  Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||183M/1.92G] Uptime: 2 days, 12:34:13
  Swp[                                                                0K/0K]

  Main  I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
  60508 root        20   0  9100  5060  3780 R   1.3  0.3  0:00.06 htop
      1 root        20   0 22828 14696 10088 S   0.0  0.7  0:34.38 /usr/lib/systemd/systemd --system --deserialize=45
    863 root        20   0  7324  2792  2664 S   0.0  0.1  0:01.01 /usr/sbin/cron -f -P
    864 messagebus  20   0  7716  4936  3912 S   0.0  0.2  0:03.67 @dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --sysl
    878 root        20   0 17408  8628  7604 S   0.0  0.4  0:01.18 /usr/lib/systemd/systemd-logind
    923 root        20   0  7184  2316  2188 S   0.0  0.1  0:00.00 /sbin/agetty -o -p -- \u --keep-baud 115200,57600,38400,9600 - vt220
    957 root        20   0  108M 22872 13400 S   0.0  1.1  0:00.25 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-si
    959 root        20   0  7140  2224  2096 S   0.0  0.1  0:00.05 /sbin/agetty -o -p -- \u --noclear - linux
    995 root        20   0  108M 22872 13400 S   0.0  1.1  0:00.00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-si
   1686 root        20   0 1201M  1216   192 S   0.0  0.1  0:00.00 /opt/digitalocean/bin/droplet-agent
   1687 root        20   0 1201M  1216   192 S   0.0  0.1  0:00.58 /opt/digitalocean/bin/droplet-agent
   1688 root        20   0 1201M  1216   192 S   0.0  0.1  0:00.00 /opt/digitalocean/bin/droplet-agent
   1689 root        20   0 1201M  1216   192 S   0.0  0.1  0:00.06 /opt/digitalocean/bin/droplet-agent
   1690 root        20   0 1201M  1216   192 S   0.0  0.1  0:00.00 /opt/digitalocean/bin/droplet-agent
   1691 root        20   0 1201M  1216   192 S   0.0  0.1  0:00.00 /opt/digitalocean/bin/droplet-agent
   1692 root        20   0 1201M  1216   192 S   0.0  0.1  0:00.06 /opt/digitalocean/bin/droplet-agent
  11456 systemd-ne  20   0 18280  9608  8456 S   0.0  0.5  0:00.28 /usr/lib/systemd/systemd-networkd
  11470 root        19  -1  132M 92840 91944 S   0.0  4.6  0:28.88 /usr/lib/systemd/systemd-journald
  12096 systemd-ti  20   0 90204  7660  6764 S   0.0  0.4  0:00.62 /usr/lib/systemd/systemd-timesyncd
  12097 systemd-ti  20   0 90204  7660  6764 S   0.0  0.4  0:00.00 /usr/lib/systemd/systemd-timesyncd
  12161 root        20   0 26960  7980  5292 S   0.0  0.4  0:00.40 /usr/lib/systemd/systemd-udevd
  12363 systemd-re  20   0 21436 13204 10772 S   0.0  0.7  0:01.72 /usr/lib/systemd/systemd-resolved
  12706 root        20   0  643M 40360 34812 S   0.0  2.0  0:04.22 /usr/libexec/fwupd/fwupd
  12716 root        20   0  643M 40360 34812 S   0.0  2.0  0:04.26 /usr/libexec/fwupd/fwupd
  12855 root        20   0  457M 13436 11388 S   0.0  0.7  0:00.12 /usr/libexec/udisks2/udisksd
  12856 root        20   0  457M 13436 11388 S   0.0  0.7  0:03.58 /usr/libexec/udisks2/udisksd
  12857 root        20   0  457M 13436 11388 S   0.0  0.7  0:00.02 /usr/libexec/udisks2/udisksd
  12859 root        20   0  457M 13436 11388 S   0.0  0.7  0:00.00 /usr/libexec/udisks2/udisksd
  12860 root        20   0  457M 13436 11388 S   0.0  0.7  0:00.00 /usr/libexec/udisks2/udisksd
  12861 root        20   0  457M 13436 11388 S   0.0  0.7  0:00.00 /usr/libexec/udisks2/udisksd
  12880 root        20   0  643M 40360 34812 S   0.0  2.0  0:00.00 /usr/libexec/fwupd/fwupd
  12881 root        20   0  643M 40360 34812 S   0.0  2.0  0:05.09 /usr/libexec/fwupd/fwupd
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 3. HTOP Command Before Pen-testing and Mitigation

Figure 4 shows the results of the *tail command cat /var/log/apache2/access.log | toilet -l*. This Figure presents data on the total requests received by the server before the mitigation and pen-testing *process* was carried out. The total was 52,562 requests by the tail command . This data serves as the basis for analyzing server conditions before protection efforts are implemented.

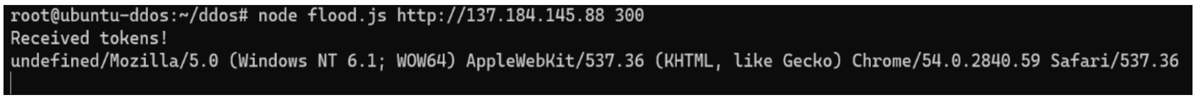


```
root@ubuntu-s-2vcpu-2gb-90gb-intel-nyc1-01:~# cat /var/log/apache2/access.log | wc -l
52562
root@ubuntu-s-2vcpu-2gb-90gb-intel-nyc1-01:~# |
```

Figure 4. Number of Requests Before Pen-testing and Mitigation

Attack Stage

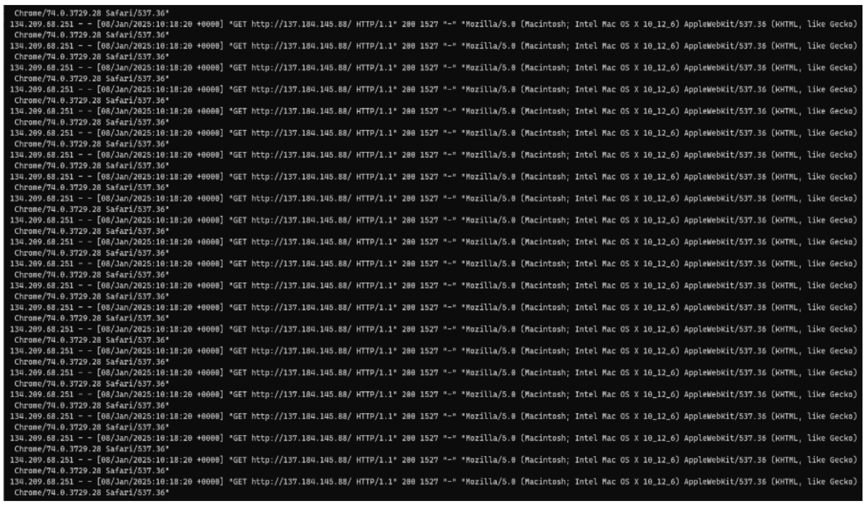
Figure 5 shows the running process of pen-testing. The analysis of Figure 5 was performed using *flood.js* tools from GitHub and ran them with a node command *flood.js <target> <time>*, with a target http://137.184.145.88/ for 600 seconds. This tool will attack the http://137.184.145.88/ for 300 seconds (5 minutes) using the user agent "Mozilla/5.0 (windows NT 6.1; WOW64) AppleWebkit/537.46 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/537.36". Furthermore, pen-testing log analysis was carried out to monitor the condition of the server when this DDoS attack occurred.



```
root@ubuntu-ddos:~/ddos# node flood.js http://137.184.145.88 300
Received tokens!
undefined/Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/537.36
|
```

Figure 5. The Process of Pen-testing

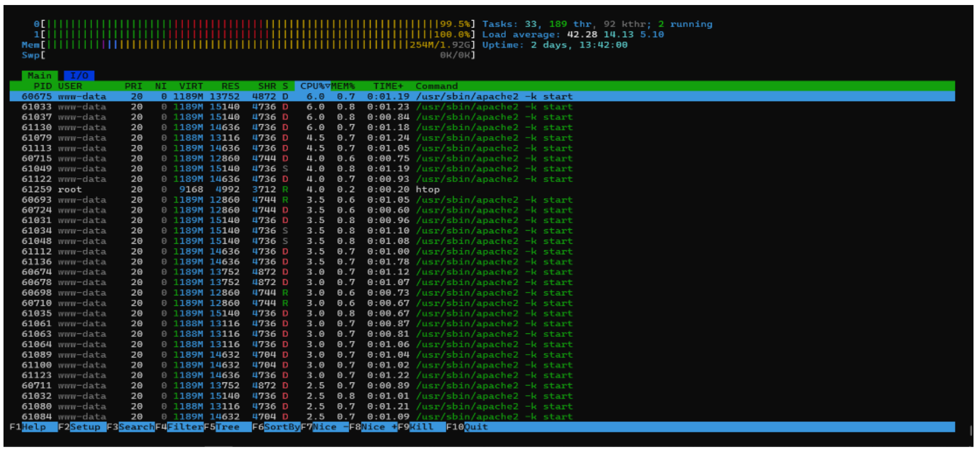
Figure 6 shows the results of the *tail command -f /var/log/apache2/access.log*. The access logs from the Apache Web Server has a large number of anomalous requests. This happens when the pen-testing process is running so that it can be concluded that this request comes from the pen-testing tool that the researcher runs. The log entry provides detailed information about an HTTP request to the server. The request originated from the IP address 137.184.188.88, which identifies the accessing client. The access occurred at the timestamp [07/Jan/2025:18:18:28 +0000], indicating the exact date and time of the request. The HTTP method used was GET, meaning the client requested to retrieve a page or resource from the server. The referenced address was http://137.184.188.88/, showing the specific URL that was accessed. The HTTP version used for this request was HTTP/1.1, a commonly used protocol version. The server responded with a status code of 200, indicating that the request was successful. The response size was 1527 bytes, representing the amount of data sent back to the client. Lastly, the User-Agent reveals that the request was made using Chrome 70 on Mac OS X 10.12.0, providing insight into the client's browser and operating system.



```
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
```

Figure 6. Real-time Logs when Pen-testing Runs

Figure 7 shows the results of the HTOP command log when the pen-testing is running. The system condition shown in Figure 7 indicates that the server is experiencing a heavy workload. The CPU usage is critically high at 99%, suggesting the processor is operating at near full capacity. Memory usage is at 254 MB out of 981 MB, which, while not excessive, still reflects increased activity. No swap memory is being utilized (0B/0B), meaning the system has not yet exceeded its physical memory limits. However, the load average is extremely high at 42.28, far exceeding the number of CPU cores, which signals a significant strain on the system.

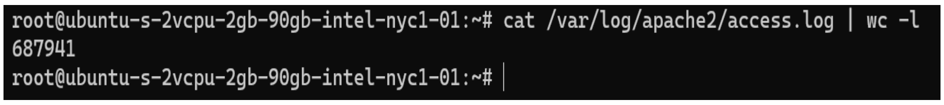


```
    0[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||99.5%] Tasks: 33, 189 thr, 92 kthr; 2 running
    1[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%] Load average: 42.28 14.13 5.10
  Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||254M/1.92G] Uptime: 2 days, 13:42:00
  Swp[                                                                  0K/0K]

  Main  I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
  60675 www-data    20   0 1189M 13752  4872 D   6.0  0.7  0:01.19 /usr/sbin/apache2 -k start
  61033 www-data    20   0 1189M 15140  4736 D   6.0  0.8  0:01.23 /usr/sbin/apache2 -k start
  61037 www-data    20   0 1189M 15140  4736 D   6.0  0.8  0:00.84 /usr/sbin/apache2 -k start
  61130 www-data    20   0 1189M 14636  4736 D   6.0  0.7  0:01.18 /usr/sbin/apache2 -k start
  61079 www-data    20   0 1188M 13116  4736 D   4.5  0.7  0:01.24 /usr/sbin/apache2 -k start
  61113 www-data    20   0 1189M 14636  4736 D   4.5  0.7  0:01.05 /usr/sbin/apache2 -k start
  60715 www-data    20   0 1189M 12860  4744 D   4.0  0.6  0:00.75 /usr/sbin/apache2 -k start
  61049 www-data    20   0 1189M 15140  4736 S   4.0  0.8  0:01.19 /usr/sbin/apache2 -k start
  61122 www-data    20   0 1189M 14636  4736 D   4.0  0.7  0:00.93 /usr/sbin/apache2 -k start
  61259 root        20   0  9168  4992  3712 R   4.0  0.2  0:00.20 htop
  60693 www-data    20   0 1189M 12860  4744 R   3.5  0.6  0:01.05 /usr/sbin/apache2 -k start
  60724 www-data    20   0 1189M 12860  4744 D   3.5  0.6  0:00.60 /usr/sbin/apache2 -k start
  61031 www-data    20   0 1189M 15140  4736 D   3.5  0.8  0:00.96 /usr/sbin/apache2 -k start
  61034 www-data    20   0 1189M 15140  4736 S   3.5  0.8  0:01.10 /usr/sbin/apache2 -k start
  61048 www-data    20   0 1189M 15140  4736 S   3.5  0.8  0:01.08 /usr/sbin/apache2 -k start
  61112 www-data    20   0 1189M 14636  4736 D   3.5  0.7  0:01.00 /usr/sbin/apache2 -k start
  61136 www-data    20   0 1189M 14636  4736 D   3.5  0.7  0:01.78 /usr/sbin/apache2 -k start
  60674 www-data    20   0 1189M 13752  4872 D   3.0  0.7  0:01.12 /usr/sbin/apache2 -k start
  60678 www-data    20   0 1189M 13752  4872 D   3.0  0.7  0:01.07 /usr/sbin/apache2 -k start
  60698 www-data    20   0 1189M 12860  4744 R   3.0  0.6  0:00.73 /usr/sbin/apache2 -k start
  60710 www-data    20   0 1189M 12860  4744 R   3.0  0.6  0:00.67 /usr/sbin/apache2 -k start
  61035 www-data    20   0 1189M 15140  4736 D   3.0  0.8  0:00.67 /usr/sbin/apache2 -k start
  61061 www-data    20   0 1188M 13116  4736 D   3.0  0.7  0:00.87 /usr/sbin/apache2 -k start
  61063 www-data    20   0 1188M 13116  4736 D   3.0  0.7  0:00.81 /usr/sbin/apache2 -k start
  61064 www-data    20   0 1188M 13116  4736 D   3.0  0.7  0:01.06 /usr/sbin/apache2 -k start
  61089 www-data    20   0 1189M 14632  4704 D   3.0  0.7  0:01.04 /usr/sbin/apache2 -k start
  61100 www-data    20   0 1189M 14632  4704 D   3.0  0.7  0:01.02 /usr/sbin/apache2 -k start
  61123 www-data    20   0 1189M 14636  4736 D   3.0  0.7  0:01.22 /usr/sbin/apache2 -k start
  60711 www-data    20   0 1189M 13752  4872 D   2.5  0.7  0:00.89 /usr/sbin/apache2 -k start
  61032 www-data    20   0 1189M 15140  4736 D   2.5  0.8  0:01.01 /usr/sbin/apache2 -k start
  61080 www-data    20   0 1188M 13116  4736 D   2.5  0.7  0:01.21 /usr/sbin/apache2 -k start
  61084 www-data    20   0 1189M 14632  4704 D   2.5  0.7  0:01.09 /usr/sbin/apache2 -k start
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 7. HTOP Logs when Pen-testing Runs

An analysis of the active processes shows that all visible processes are associated with the Apache Web Server, with most of them being run by the www-data user, which is the default user for Apache operations. Although individual CPU usage per process remains relatively low, the sheer number of active Apache processes contributes to the overall CPU saturation. Additionally, the repetition of the command "/usr/sbin/apache2 -k start" confirms the presence of multiple concurrent Apache instances, further indicating that the server is likely under an HTTP-based attack or handling an unusually high number of simultaneous requests. Based on these conditions, the server is experiencing serious performance problems, namely the average load has increased drastically (from 9.18 to 42.28) and the upward trend of load averages indicates deteriorating performance problems.

Figure 8 shows the number of requests when pen-testing runs. This figure presents a spike in requests to the server during pen-testing. It can be seen that the surge increased drastically from 52,562 (Figure 4) to 687,941 requests in five minutes. These spikes overload the server and risk causing errors if they last for a long time.

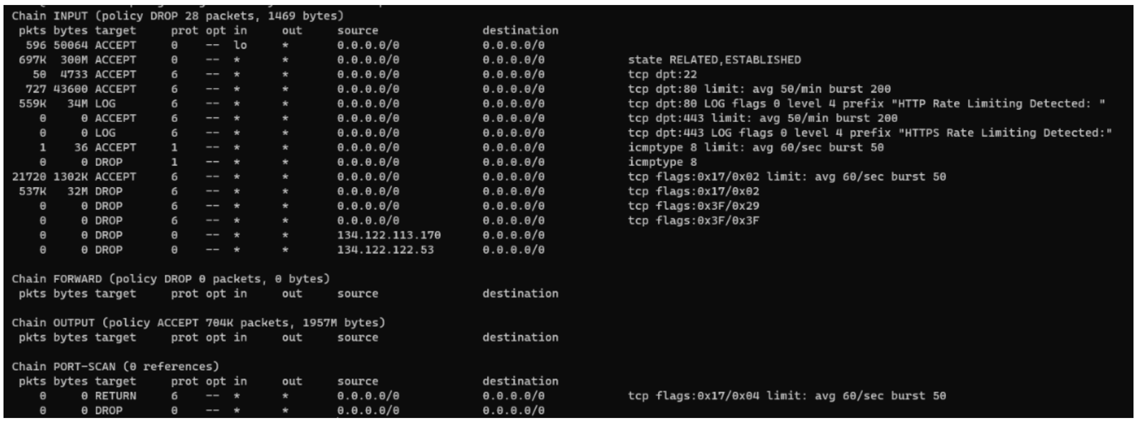


```
root@ubuntu-s-2vcpu-2gb-90gb-intel-nyc1-01:~# cat /var/log/apache2/access.log | wc -l
687941
root@ubuntu-s-2vcpu-2gb-90gb-intel-nyc1-01:~# |
```

Figure 8. Number of Requests when Pen-testing Runs

Mitigation Stage

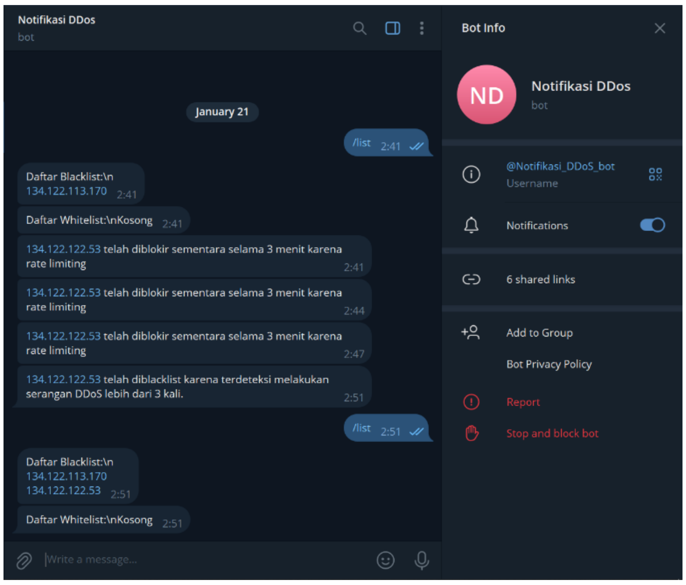
The application of rate limiting through iptables can be observed by running the command sudo iptables -L-v-n. The display results of the command is presented in Figure 9.



```
Chain INPUT (policy DROP 28 packets, 1469 bytes)
 pkts bytes target     prot opt in     out     source               destination
  596 50064 ACCEPT     0    --  lo     *       0.0.0.0/0            0.0.0.0/0
 697K  300M ACCEPT     0    --  *      *       0.0.0.0/0            0.0.0.0/0            state RELATED,ESTABLISHED
   50  4733 ACCEPT     6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:22
  727 43600 ACCEPT     6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:80 limit: avg 50/min burst 200
 559K   34M LOG        6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:80 LOG flags 0 level 4 prefix "HTTP Rate Limiting Detected: "
    0     0 ACCEPT     6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:443 limit: avg 50/min burst 200
    0     0 LOG        6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp dpt:443 LOG flags 0 level 4 prefix "HTTPS Rate Limiting Detected:"
    1    36 ACCEPT     1    --  *      *       0.0.0.0/0            0.0.0.0/0            icmptype 8 limit: avg 60/sec burst 50
    0     0 DROP       1    --  *      *       0.0.0.0/0            0.0.0.0/0            icmptype 8
21720 1302K ACCEPT     6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x17/0x02 limit: avg 60/sec burst 50
 537K   32M DROP       6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x17/0x02
    0     0 DROP       6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x3F/0x29
    0     0 DROP       6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x3F/0x3F
    0     0 DROP       0    --  *      *       134.122.113.170      0.0.0.0/0
    0     0 DROP       0    --  *      *       134.122.122.53       0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 704K packets, 1957M bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain PORT-SCAN (0 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     6    --  *      *       0.0.0.0/0            0.0.0.0/0            tcp flags:0x17/0x04 limit: avg 60/sec burst 50
    0     0 DROP       0    --  *      *       0.0.0.0/0            0.0.0.0/0
```

Figure 9. IP Tables Mitigation Results

Figure 9 shows how the rules applied in the bash script *(.sh)* appear. The bash script is used to automatically configure firewall rules and apply rate limiting to mitigate potential HTTP flood attacks. In this log it is also seen that IP 134.122.113.170 and IP 134.122.122.53 are blocked just as they happened inside the Telegram bot that is presentes in Figure 10.



/list 2:41

Daftar Blacklist:\n
134.122.113.170 2:41

Daftar Whitelist:\nKosong 2:41

134.122.122.53 telah diblokir sementara selama 3 menit karena rate limiting 2:41

134.122.122.53 telah diblokir sementara selama 3 menit karena rate limiting 2:44

134.122.122.53 telah diblokir sementara selama 3 menit karena rate limiting 2:47

134.122.122.53 telah diblacklist karena terdeteksi melakukan serangan DDoS lebih dari 3 kali. 2:51

/list 2:51

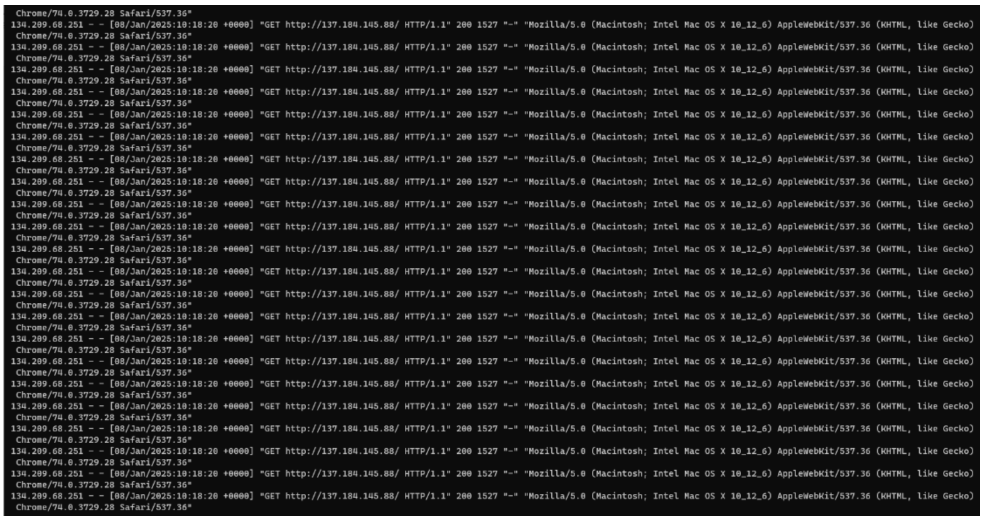
Daftar Blacklist:\n
134.122.113.170
134.122.122.53 2:51

Daftar Whitelist:\nKosong 2:51

Figure 10. Telegram Bot Display

Figure 10 also shows that any IP that tries to attack the script will send a notification to the bot in real-time. Each IP will have 3 chances until it is blocked. Any IP that is blocked cannot access the server until the blacklist is removed by typing *command/unblacklist*. *Command/list* also has the ability to bring up any IPs that are being blacklisted.

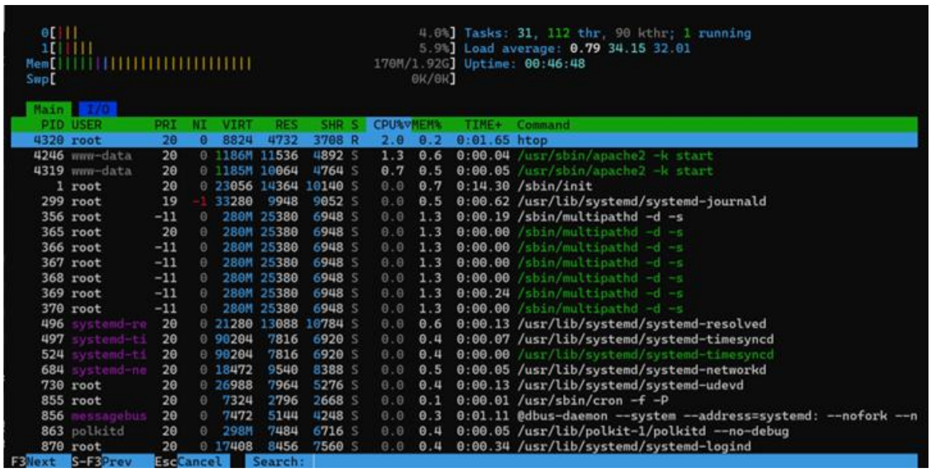
After mitigating using rate limiting, the researcher also looked at the log information generated. The log information recorded after the mitigation implementation is presented in Figure 11. It can be seen that there is still an anomalous request from IP 134.209.68.251. However, this is a reasonable request because the request will still be accepted even if it is limited by a rate limiting of 50 requests to a burst of 200 requests. During this test, the demand was not as massive as when pen-testing was done before the rate limiting mitigation was created.



```
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
134.209.68.251 - - [08/Jan/2025:10:18:20 +0000] "GET http://137.184.145.88/ HTTP/1.1" 200 1527 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/74.0.3729.28 Safari/537.36"
```

Figure 11. Real-time Request Logs on Mitigation

The system condition presented in Figure 12 indicates a stable and low-load state at the time of observation. The memory usage is 170 MB out of 981 MB, and no swap memory is being used (0B/0B), suggesting efficient memory performance. The CPU load is currently low, indicating the system is operating in a normal or stable condition. Each process is consuming a relatively small amount of memory, further confirming light resource usage.



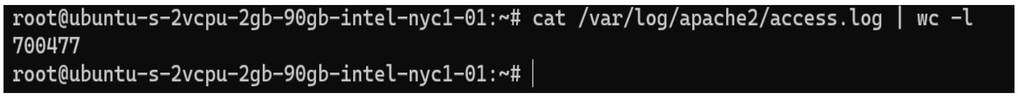
```
    0[|||                                   4.0%] Tasks: 31, 112 thr, 90 kthr; 1 running
    1[|||||                                 5.9%] Load average: 0.79 34.15 32.01
  Mem[||||||||||||||||||||||||        170M/1.92G] Uptime: 00:46:48
  Swp[                                   0K/0K]

  Main  I/O
    PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
   4320 root        20   0  8824  4732  3708 R   2.0  0.2  0:01.65 htop
   4246 www-data    20   0 1186M 11536  4892 S   1.3  0.6  0:00.04 /usr/sbin/apache2 -k start
   4319 www-data    20   0 1185M 10064  4764 S   0.7  0.5  0:00.05 /usr/sbin/apache2 -k start
      1 root        20   0 23056 14364 10140 S   0.0  0.7  0:14.30 /sbin/init
    299 root        19  -1 33280  9948  9052 S   0.0  0.5  0:00.62 /usr/lib/systemd/systemd-journald
    356 root       -11   0  280M 25380  6948 S   0.0  1.3  0:00.19 /sbin/multipathd -d -s
    365 root        20   0  280M 25380  6948 S   0.0  1.3  0:00.00 /sbin/multipathd -d -s
    366 root       -11   0  280M 25380  6948 S   0.0  1.3  0:00.00 /sbin/multipathd -d -s
    367 root       -11   0  280M 25380  6948 S   0.0  1.3  0:00.00 /sbin/multipathd -d -s
    368 root       -11   0  280M 25380  6948 S   0.0  1.3  0:00.00 /sbin/multipathd -d -s
    369 root       -11   0  280M 25380  6948 S   0.0  1.3  0:00.24 /sbin/multipathd -d -s
    370 root       -11   0  280M 25380  6948 S   0.0  1.3  0:00.00 /sbin/multipathd -d -s
    496 systemd-re  20   0 21280 13088 10784 S   0.0  0.6  0:00.13 /usr/lib/systemd/systemd-resolved
    497 systemd-ti  20   0 90204  7816  6920 S   0.0  0.4  0:00.07 /usr/lib/systemd/systemd-timesyncd
    524 systemd-ti  20   0 90204  7816  6920 S   0.0  0.4  0:00.00 /usr/lib/systemd/systemd-timesyncd
    684 systemd-ne  20   0 18472  9540  8388 S   0.0  0.5  0:00.05 /usr/lib/systemd/systemd-networkd
    730 root        20   0 26988  7964  5276 S   0.0  0.4  0:00.13 /usr/lib/systemd/systemd-udevd
    855 root        20   0  7324  2796  2668 S   0.0  0.1  0:00.01 /usr/sbin/cron -f -P
    856 messagebus  20   0  7472  5144  4248 S   0.0  0.3  0:01.11 @dbus-daemon --system --address=systemd: --nofork --n
    863 polkitd     20   0  298M  7484  6716 S   0.0  0.4  0:00.05 /usr/lib/polkit-1/polkitd --no-debug
    870 root        20   0 17408  8456  7560 S   0.0  0.4  0:00.34 /usr/lib/systemd/systemd-logind
F3Next  S-F3Prev  EscCancel   Search:
```

Figure 12. HTOP Logs when Mitigation is Executed

The load average over the last 1 minute is 0.79, which is considered normal, while the 5-minute and 15-minute load averages are significantly higher at 34.15 and 32.01, respectively. This suggests that the system was previously experiencing a high processing load, possibly due to an attack or intensive workload, but has since returned to a stable state.

Based on these statistics, we can conclude that The system was recently under heavy load, as reflected by the high 5- and 15-minute averages. The current condition has normalized, shown by the low 1-minute load average of 0.79. Furthermore, the total number of processes is relatively low (31), implying that this is likely a dedicated server with specific functions rather than a general-purpose host.

Figure 13 shows the number of requests after mitigation. Although the number of requests is very large, it should be noted that the previous number of requests is 700477 according to Figure 8. So the number of requests received by the server when mitigation has been carried out is 700477 - 687941 = 12,536. This number is very small when compared to the number of requests before mitigation, which amounted to 635,379 requests.



```
root@ubuntu-s-2vcpu-2gb-90gb-intel-nyc1-01:~# cat /var/log/apache2/access.log | wc -l
700477
root@ubuntu-s-2vcpu-2gb-90gb-intel-nyc1-01:~# |
```

Figure 13. Number of Requests After Mitigation

With a bash script organized into iptables, this study established a rule that the requests sent were limited to only 50 requests and bursts to 200 requests. Through the rules that have been applied, research results have been successfully obtained with significant value. Table 2 shows the server performance before and after mitigation, The CPU utilization before mitigation reached 100%, but after mitigation it was reduced to below 7%. In addition, the number of requests received by the server was also very significant, from 635,379 requests before mitigation to only 12,536 requests after mitigation.

Table 2. Server Performance Before and After Mitigation

No.	Component	Before Mitigation	After Mitigation
1.	CPU 1	100 %	4.0 %
2.	CPU 2	100 %	5.9%
3.	Memory	254 MB	170 Mb
4.	Demand	635.379	12,536

CONCLUTION AND SUGGESTIONS

This study successfully analyzed the HTTP GET DDoS Flood attack and its mitigation techniques. Mitigation methods using a well-configured firewall have proven effective. The right firewall rules can improve server performance. The CPU load was successfully suppressed during the attack. The rate limiting technique also plays an important role in limiting the number of requests to the server. This helps prevent the server from going down. In addition, the real-time notification system through the Telegram bot is very useful in providing early warnings when an attack occurs and can instantly remove errors that appear in the system. The suggestion for further research is to use a combination of several mitigation techniques, such as using CDN services as an additional option. Additionally, it is important for web administrators to regularly update and configure security systems to remain effective against new threats.

BIBLIOGRAPHY

[1] F. A. A. Putra, A. R. Jatmiko, R. M. A. Arief, and M. I. A. Ardiansa, "Rancang Bangun Sistem Informasi Kepegawaian dan Inventaris Di Universitas Merdeka Malang Berbasis Web Menggunakan Framework Codeigniter," *Jurnal RESTIKOM : Riset Teknik Informatika dan Komputer*, vol. 5, no. 2, 2023, doi: 10.52005/restikom.v5i2.149.

[2] F. Hamdani, Y. B. Fitriana, and N. Oper, "Analisis Keamanan Website Terhadap Serangan DDOS Menggunakan Metode National Institute of Standards and Technology (NIST)," *KLIK: Kajian Ilmiah Informatika dan Komputer*, vol. 3, no. 6, 2023.

[3] N. Mamuriyah, S. E. Prasetyo, and A. O. Sijabat, "Rancangan Sistem Keamanan Jaringan dari serangan DDoS Menggunakan Metode Pengujian Penetrasi," *Jurnal Teknologi Dan Sistem Informasi Bisnis*, vol. 6, no. 1, 2024, doi: 10.47233/jteksis.v6i1.1124.

[4] E. Nofarita, "Implementasi Aplikasi Software Natural Network Mendeteksi Tingkatan Serangan DDoS Pada Jaringan Komputer," *Elkom : Jurnal Elektronika dan Komputer*, vol. 14, no. 2, 2021, doi: 10.51903/elkom.v14i2.501.

[5] M. N. Faiz, O. Somantri, and A. W. Muhammad, "Rekayasa Fitur Berbasis Machine Learning untuk Mendeteksi Serangan DDoS," *Jurnal Nasional Teknik Elektro dan Teknologi Informasi*, vol. 11, no. 3, 2022, doi: 10.22146/jnteti.v11i3.3423.

[6] S. Park, Y. Kim, H. Choi, Y. Kyung, and J. Park, "HTTP DDoS flooding attack mitigation in software-defined networking," *IEICE Trans Inf Syst*, vol. E104D, no. 9, 2021, doi: 10.1587/transinf.2021EDL8022.

[7] N. Sugianti, Y. Galuh, S. Fatia, and K. F. H. Holle, "Deteksi Serangan Distributed Denia of Services (DDOS) Berbasis HTTP Menggunakan Metode Fuzzy Sugeno," *JISKA (Jurnal Informatika Sunan Kalijaga)*, vol. 4, no. 3, 2020, doi: 10.14421/jiska.2020.43-03.

[8] F. Nisa and S. Ramadona, "Sistem Pencegahan Serangan Distributed Denial Of Service Pada Jaringan SDN," *Jurnal Sistim Informasi dan Teknologi*, vol. 5, no. 3, 2023.

[9] J. Hansen and T. Sutabri, "Mendesain Cyber Security Untuk Mencegah Serangan DDoS Pada Website Menggunakan Metode Captcha," *Digital Transformation Technology*, vol. 3, no. 1, 2023.

[10] D. Firdaus, I. Sumardi, and G. Nugraha, "Peningkatan Keamanan Server GraphQL Terhadap Serangan DDOS Dengan Tipe Batch Attack Menggunakan Metode Rate Limiting," *Cyber Security dan Forensik Digital*, vol. 7, no. 2, pp. 62–68, 2024, doi: 10.14421/csecurity.2024.7.2.4718.

[11] A. El Kamel, "A GNN-Based Rate Limiting Framework for DDoS Attack Mitigation in Multi-Controller SDN," in *Proceedings - IEEE Symposium on Computers and Communications*, 2023. doi: 10.1109/ISCC58397.2023.10218204.

[12] M. T. A. Zaen, A. Tantoni, and M. Ashari, "DDoS Attack Mitigation With Intrusion Detection System (IDS) Using Telegram Bots," *JISA(Jurnal Informatika dan Sains)*, vol. 4, no. 2, 2021, doi: 10.31326/jisa.v4i2.1043.

[13] I. D. Wiradyaksa, D. H. Putri, R. M. Iqbal, N. H. Astari, N. Karna, and F. Dewanta, "Design and Implementation of Automated Web Application Firewall, Rate Limiting, and Intrusion Detection System for Cyber Defense," in *2024 8th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, IEEE, Aug. 2024, pp. 256–261. doi: 10.1109/ICITISEE63424.2024.10730693.

[14] Y. Arta, R. Wandri, A. Hanafiah, B. K. Pranoto, and M. R. Fadhilah, "Analisa Perbandingan Web Server Untuk Kebutuhan Open Journal System (OJS) Menggunakan Secure Tunnel," *CogITo Smart Journal*, vol. 8, no. 2, 2022, doi: 10.31154/cogito.v8i2.407.537-548.

[15] G. Fanani and I. Riadi, "Analysis of Digital Evidence on Denial of Service (DoS) Attack Log Based," *Buletin Ilmiah Sarjana Teknik Elektro*, vol. 2, no. 2, 2020, doi: 10.12928/biste.v2i2.1065.

[16] R. I. P. Siagian, F. A. Lubis, M. A. Syuja, and D. Kiswanto, "Analisis Performa Sistem Operasi Manjaro Linux Dalam Lingkungan Komputasi Desktop Virtual," *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 9, no. 1, pp. 1266–1272, 2025, doi: 10.36040/jati.v9i1.12668.

[17] B. Jaya, Y. Yuhandri, and S. Sumijan, "Peningkatan Keamanan Router Mikrotik Terhadap Serangan Denial of Service (DoS)," *Jurnal Sistim Informasi dan Teknologi*, 2020, doi: 10.37034/jsisfotek.v2i4.32.