ISSN 2278-3091
Volume 8, No.4, July – August 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse86842019.pdf
https://doi.org/10.30534/ijatcse/2019/86842019

Enhanced Detection Algorithms to Detect HTTP DDoS

Ghafar A. Jaafar[1], Shahidan M. Abdullah[2], Saiful Adli [3]
[1,2,3] Razak Faculty of Technology and Informatics,
Universiti Teknologi Malaysia (UTM) 54100 Kuala Lumpur, Malaysia.
[1] afastars@gmail.com
[2] mshahidan@utm.my
[3] saifuladli@utm.my

ABSTRACT

A web application utilizes Hypertext Transfer Protocol (HTTP) to surf client requests. This protocol is used widely, especially in business areas such as in online transactions and websites, including in government websites. A client delivers information to a server carried by a client web browser. An HTTP distributed denial of service (DDoS) attack occurs when the attacker is able to mimic client information, which makes a DDoS attack at the application layer difficult to distinguish as the traffic pattern is similar to a genuine request. Furthermore, it is not compulsory for the client to present the GET headers component to a web server during the GET request transaction. Existing detection of HTTP DDoS attacks still faces challenges in differentiating between authentic and bogus GET requests in real time. In this paper, a fast algorithm (FARGO) method to detect HTTP DDoS attacks is introduced. FARGO consists of three detection algorithms to recognize HTTP DDoS categories as request flooding attacks. The assessment of the proposed detection system was conducted in real experimental conditions with real attack scripts. The proposed detection method provided expected outcomes with improvements of 11.30% for true positive rates and 8.9% for false-positive rates.

Key words: DoS, DDoS, Denial of Service attack, Application layer DDoS

1. INTRODUCTION

A web server utilizes HTTP and HTTPS protocols to surf client requests through a network. These protocols are widely used in many business areas such as in payment gateways, online purchasing, banking, credit card transactions, etc. Attackers target these services by causing a web server to become unavailable through launching attacks known as HTTP DDoS attacks. Most DDoS attacks executed at the application layer nowadays target the HTTP port [1]. DDoS attacks lead to loss of trust and revenue [2]. Past studies [2-4] have found DDoS attack patterns at the application layer as being similar to genuine request packets. Singh, Singh [5] attributed the existence of DDoS attacks to application settings and functions that cause the attack to target web server resources such as the CPU, network and memory [6].

DDoS attacks execute at layer seven requires establishing TCP connection to a web server before can perform a transaction at the application layer. Single TCP connection can contain multiple GET request [7] due to this circumstance attacker sent the GET request continuously. To generate massive traffic, the attacker utilizes botnets to deliver a high number of requests from a web server. The location of botnets is also distributed, which makes the generation of enormous amounts of traffic easy and fast. Aamir and Zaidi [8] explained that due to botnet architecture being scattered and comprised of a number of compromised machines, a DoS attack can easily turn into a DDoS attack.

Nowadays, modern devices adopt the Internet of Things technology, which refers to everything that can access the Internet, such as a television set. The Internet of Things (IoT) opens up a new avenue for an attacker to launch DDoS attacks, which will contribute to more cyber-attacks. Devices with IoT technology can become easily accessible tools in a cyber army's plan to launch attacks as they are always connected to the Internet with a naive security update and patch [9]. The emerging of generation five networks (5G) deliver a significant impact to formation of HTTP DDoS attack. The 5G network provides faster speed and more reliable connection for other equipment's and smart phone[10]. HTTP DDoS can be launched in many platforms such as IOT devices, which comprise of smart phone and cameras, which can utilize 5G network. These types of devices are vulnerable to be affected due to week security patch. Launch HTTP DDoS by adopted 5G network lead to faster flooding of the web server due to higher speed of the Internet provided by the network.

Aside from that, the existence of online services, such as boosters or stressers, to easily facilitate such attacks contributes to the increase in such attacks [11]. A DDoS attack is easier to implement than other network attacks as it only requires a large number of zombie machines and minimal knowledge of security to execute the attack [12]. By using an efficient attack script, the attacker requires fewer resources to generate huge amounts of traffic [13, 14].

Several detection methods have been introduced to recognize such attacks. Nevertheless, due to their ability to mimic normal client behavior, and the different strategies devised to launch attacks, these attacks are difficult to recognize. In this paper, a method of detecting HTTP DDoS attacks, focusing on flooding attacks, is presented. The proposed detection method is called FARGO and consists of three detection algorithms segregated into regular inspection, medium inspection and deep inspection. Each algorithm consists of GET header components that have been used by authentic clients to browse the content of a web server.

The proposed detection technique allowed an administrator to select which inspection they wanted in order to recognize the attack. Regular inspection determines whether the source of the GET request was initiated from a web browser or via automated tools by accessing browser width and height. Medium inspection combines regular inspection with checking the existence of GET headers such as user-agent, accept-language, connection and accept-encoding. Deep inspection combines regular, medium and query inspection received in GET requests. The selection of GET components was based on traffic analysis of components that were commonly employed by attackers to make the traffic appear genuine. Results obtained from the experiments (Section 7) indicated that the detection method worked as expected where malicious GET requests instituted by HTTP DDoS attacks were detected successfully. The main contributions of this paper can be summarized as follows:

i) Introduction of three detection algorithms to detect HTTP DDoS attacks.
ii) Increasing true positive rates and reducing false positive rates.
iii) The proposed detection method can be implemented in production network environments.

The rest of the paper is organized as follows. Section 2 explains DDoS attack categories at the application layer. Section 3 provides an overview of detection systems proposed by prior studies. Section 4 explains the experiment architecture and datasets. Section 5 describes about traffic analysis and feature extraction. Section 6 elaborates about the analysis command. Dataset analysis conducted in Section 7, while Section 8 explained about the proposed detection algorithms while section 9 provides details about evaluation architecture. Result and discussion in section 10 and continue with section 11 comparison with other studies. A summary and future works are described in Section 12.

2. HTTP DDOS ATTACK CATEGORIES

DDoS attacks at the application layer are divided into several categories, as explained by previous researchers [15-20]. Detailed explanations pertaining to the categories are as follows:

2.1 Session Flooding Attack: Server resources become overloaded due to enormous session request rates. This type of attack is known as DDoS flooding and the attacker requires a large number of genuine HTTP requests. The attacker usually utilizes a botnet because it has the ability to launch a valid request. Commonly, GET requests initiated by a botnet exceed 10 requests per second.

2.2 Request Flooding Attack: This attack category takes advantage of HTTP 1.1 structure, which allows multiple requests in one session. Due to this vulnerability, attackers generate vast numbers of requests in one session, which are larger than genuine users' requests. Rai and Challa [21] stated that the attacks utilized botnets to launch the attack. This type of attack is able to cause a server to become unresponsive when it receives the huge number of HTTP GET requests generated by a botnet.

2.3 Asymmetric Attack: A server experiences a high workload when an attacker employs an HTTP session to send a high number of requests to download files from a database server, or excessive requests to execute queries from the server.

2.4 Slow Request/Response Attack: Partial HTTP requests are sent, which grow quickly, slowly update and never close the connection, which makes the server socket unavailable. The attack operates under the threshold limit and with traffic patterns similar to authentic patterns, which make it difficult to detect [21]. A server will continuously remind clients to open its connections as each GET request received fails to complete the transaction, which will result in clients being unable to gain access to a server due to full concurrent connections.

3. RELATED WORKS

In this paper, HTTP GET header components are utilized as a form of detection to recognize DDoS attacks executed at the application layer, in order to improve true positive rates and false positive rates. Many prior studies have introduced solutions pertaining to such attacks. This section summarizes the current research work on detecting HTTP DDoS attacks.

Hameed and Ali [11] introduced a DDoS detection method that executes at the network and application layers. The detection method consists of two components called the capture server and detection server. The detection server will receive live network traffic from a capture server for processing. The detection method counts incoming packets and will detect them as attacks when the source connection exceeds a pre-defined threshold responsible for capturing live traffic. The authors used timestamps, source IPs, destination IPs, packet protocols and packet headers to constitute detection.

Idhammad, Afdel [22] proposed information theoretic entropy and machine learning to detect HTTP DDoS attacks in cloud services. The detection technique proposed by the authors consists of three steps: entropy estimation, pre-processing and classification. Features of the network headers for inbound network traffic are used to estimate entropy, which is performed by an algorithm called the time-based sliding window. A pre-processing algorithm will execute when the entropy exceeds normal range. Clarification of HTTP DDoS attacks will occur at this stage, and network traffic will be cleaned and normalized to recent time windows.

Behal, Kumar [23] introduced a flexible, automated and collaborative defense system called D-FACE to detect and mitigate impacts of DDoS attacks. The idea behind this defense strategy is to compute the information of the distance between authentic and network traffic anomalies by employing information theory-based divergence metrics, to recognize types of DDoS attacks. The proposed detection technique has a minimal overhead for computational speed and memory compared with other types of deployment such as a victim-end defense.

Singh, Singh [24] employed a machine learning method to recognize HTTP DDoS attacks. The proposed detection method is able to differentiate botnets from authentic users in malicious traffic, legitimate user traffic and flash crowd traffic. The detection method is also able to identify botnet locations and assess client attitudes to detect attack traffic towards a web server. The detection technique employs request index, response index, popularity index, repetition index and classifier algorithms in order to examine user behavior and is deployed as a proxy. Meanwhile, Zhao, Zhang [25] introduced a detection method based on user access behavior characteristics. URL access pattern is used to detect DDoS attack at application layer.

Sreeram and Vuppala [26] recommended fast and early detection to recognize HTTP DDoS attacks by using a machine-learning matrix. Instead of sessions from users and packet patterns, the authors utilized time intervals to constitute a detection algorithm. Maximum sessions for one-time intervals are processed by the machine-learning matrix to detect DDoS attacks at the application layer. The authors also counted the frequency of web pages being browsed and the time gaps between first-page access and second-page access by the user to evaluate client access patterns.

Aborujilah and Musa [27] introduced a detection technique based on behavior and proposed two training and testing algorithms to identify different categories of HTTP flooding attacks. The authors utilized TCP packet headers and statistical approaches with a covariance matrix to detect HTTP DDoS attacks in the cloud environment. Normal access patterns are constructed by the training algorithm while types of traffic received are identified by the testing algorithm.

Singh and De [28] employed multilayer perceptron with a genetic algorithm (MLP-GA) to detect HTTP DDoS attacks. The detection system utilizes the number of HTTP counts, the number of IP addresses, the constant mapping function and fixed frame lengths. A GET request received by a web server will be counted and any IP address accessing a web server for more than 20 seconds will be evaluated. According to the authors, attackers utilize static protocol lengths, hence fixed frame lengths were used to conduct detection. The authors used three datasets: EPA-HTTP, CAIDA 2007 and Experiment Dataset.

Hoque, Kashyap [29] utilized correlation measures to detect real-time DDoS attacks at the victim-end. The detection approach extracts three features during pre-processing of network traffic, i.e., entropy of source IPs, variation index of source IPs and packet rate, to create a normal profile. The authors explained that the proposed system would detect attacks when the distance between normal and live traffic was more than the threshold value. The study used three datasets: CAIDA, TUIDS and DARPA.

Liao, Li [30] utilized request interval sequences and request frequency sequences to develop a detection method based on user access frequencies. According to the authors, time interval for authentic user will be longer when visiting interesting pages. However, time interval for DDoS is much shorter. The studies utilized ClarkNet HTTP and Experiment Dataset.

A client will deliver several GET header requests to a web server during the transaction of an HTTP GET request. The GET headers contain several components such as user-agent, accept-language, connection status, query, accept charset and any related header available from the client. Most of the HTTP DDoS attack detection approaches proposed in the literature exhibited minimal inspection of these components during the operation of GET requests. DDoS attacks at the application layer deliver incorrect GET headers and provide false GET header values to mimic authentic requests, in order to conceal their activity. Additionally, the use of automated tools to generate a large amount of requests is one of the minimal approaches to detection in prior studies. It is proposed here that inspection of GET header components and browser features will provide fast detection, i.e., before the attack causes a web server to become unresponsive due to enormous amounts of GET requests.

4. EXPERIMENTAL ENVIRONMENT FOR DATASET PREPARATION

This section explains the experimental equipment used to evaluate the proposed detection method and the self-generated datasets for the purpose of analysis of HTTP DDoS traffic. Due to the unavailability of datasets for HTTP DDoS attacks, this research executed real experiments to self-generate HTTP DDoS attack datasets to analyze the attack patterns. The existence of datasets for HTTP DDoS attacks has been

mentioned in several past studies. The Jazi, Gonzalez [31] limitation to gain datasets led to all prior studies utilizing simulation software like NS2 and MATLAB. Existing datasets for DDoS attacks only capture network layer information while concealing application layer information [32]. Jaafar, Abdullah [33] used real HTTP DDoS tools to predict future HTTP DDoS attack strategy and for input on recent attack patterns. Past studies adopt obsolete dataset for validation hence generate the dataset close to actual network topologies is necessary [34].

The experimental architecture consisted of a web server running on Windows 2012 R2 while the client and attacker machines were run as virtual machines. The attacker's operating system was Ubuntu while the authentic client used Windows 8. A simple HTML page was designed and run as HTTP protocol for the genuine client to access and for the attacker to launch the attack against. Four attack scripts were then selected to launch attacks against the web server. The tools used to launch the HTTP DDoS attacks were publicly available. Table 1 provides a summary.

Table 1: Experiment Components

Web Server	Client / Attack Machine	Attack Scripts	Attack Duration	Target URL
Intel(R) Core (TM) i7-6700 CPU @ 3.40GHz, 8GB Memory	Intel(R) Core (TM) i7-3770 CPU @ 3.40GHz 12GB / 8GB Memory	Golden Eye.py Blackhorizon.py ChiHulk.py HOIC.exe	5 Minutes	http://lab.com.my

5. TRAFFIC ANALYSIS AND FEATURE EXTRACTION

An analysis of HTTP DDoS attack traffic was conducted utilizing four real DDoS tools executed at the application layer. This analysis mainly focused on GET header components, with each tool being launched separately to acquire traffic patterns. DDoS attacks launched at the application layer are able to create a large number of GET requests with the help of efficient attack scripts and a minimal number of resources [13, 14]. The attack duration was specified as five minutes. Figure 1 provides a graphical view of the attack analysis architecture.

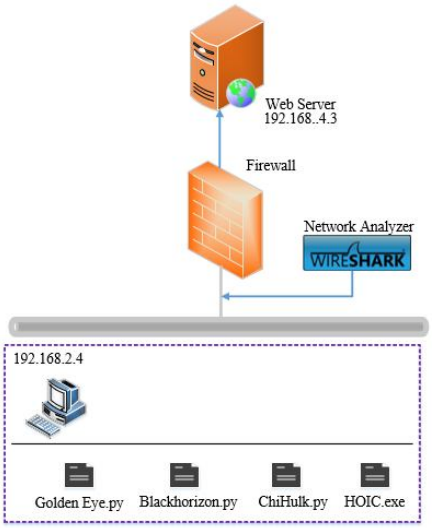

Figure 1: Analysis Environment

An HTTP DDoS attack has the ability to mimic human requests. Due to this characteristic, a number of attack strategies were utilized to mimic user access patterns. Nonetheless, for DDoS flooding attacks at the application layer, the high-frequency access pattern was equal for all attack scripts. To overwhelm web server resources immediately, the attack must be sent in high frequency. Although attacks of 5-minute duration were launched against the web server, the attack script had the capability to send large amounts of traffic against the web server. All attack scripts utilized in this paper generated thousands of requests for GET requests. A summary of generated GET requests is shown in Table 2.

Table 2: Total Traffic Generated by HTTP DDoS Attacks

No.	Attack script	Number of GET Requests
1.	ChiHulk.py	88,769
2.	Golden Eye.py	21,257
3.	Blackhorizon.py	23,974
4.	HOIC.exe	97,332

6. HTTP COMMANDS ANALYSIS

Traffic analysis of web applications requires several commands to observe communications between the client and the web server. A type of software known as Wireshark provides many commands to analyze HTTP GET headers. To facilitate the analysis process, this research outlined several commands required to execute to examine DDoS traffic patterns at the application layer. Table 3 presents the commands.

Table 3: Analysis Commands

No	Command	Details
1.	http	Shows protocol HTTP only
2.	http.request	Displays HTTP GET request
3.	http.user_agent	Displays user agent in GET request
4.	http.referrer	Displays HTTP referrer in GET request
5.	http.request.uri.query	Displays HTTP query in GET request
9.	http.connection	Shows HTTP Status in GET request

7. DATASET ANALYSIS RESULTS

The analysis conducted on all attack scripts found that a variety of user-agents were employed, which showed that a web server was accessible from a different machine without it observing the source IP address of the GET request. The attack traffic also generated query strings that could not be understood by humans, with a combination of upper-case letters, lower case letters and special characters. In addition, the source of the HTTP referral came from a valid resource with a combination of queries that consisted of numbers and characters. Table 4 presents the attack logs from the HTTP GET requests.

Table 4: Attack Logs

User Agent String	Request Query	HTTP Referrer
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Win64; x64; Trident/4.0)	KEAWOCO=ZFSUSO	http://filehippo.com/search?q=\221y\203\231\213{\214\217\222\215\r\n
Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)	QJCQABP=MIGMQXRML	http://taginfo.openstreetmap.org/search?q=~\177\235z\227\236\r\n
Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.9.1.3) Gecko/20090824 Firefox/3.5.3 (.NET CLR 3.5.30729)	QJCQABP=MIGMQXRML	http://www.baoxaydung.com.vn/news/vn/search&q=\225\211\224\235|\240\227\215\216\r\n
Mozilla/4.0 (compatible; MSIE 6.1; Windows XP)	DYH=GFOUW	https://steamcommunity.com/market/search?q=\217x\205\203\2
		26\235{\r\n
Opera/9.80 (Windows NT 5.2; U; ru) Presto/2.5.22 Version/10.51	GQHCIZNYO=ZHILUY	https://www.npmjs.com/search?q=\212\205\207x}\220\232\217\217\r\n
Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)	\357\277\275\357\277\275{\357\277\275\177=\357\277\275\357\277\275y	
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/532.1 (KHTML, like Gecko) Chrome/4.0.219.6 Safari/532.1	\357\277\275\357\277\275\357\277\275=\357\277\275\357\277\275\357\277\275	
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.1) Gecko/20090718 Firefox/3.5.1	\357\277\275\357\277\275{\357\277\275\177=\357\277\275\357\277\275y	

Detailed analysis found that the DDoS attacks launched at the application layer presented inconsistent GET headers, where some of the GET headers were missing during the transaction of the GET requests. According to [35], common GET headers utilized are Host, Connection, Accept-Encoding, User-Agent, Accept-Language, Accept and Content-Type. Besides these common headers, GET headers also contain a Keep-Alive header. Keep-Alive is a component for HTTP response that indicates attackers accidentally assigning that header to GET requests, which shows evidence that the HTTP GET request is malicious. Figure X presents the HTTP logs. Figure 2 shows Keep-Alive in the GET headers component.

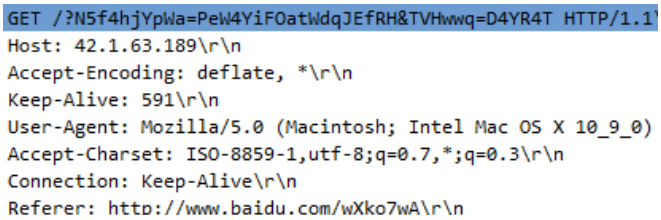

Figure 2: Keep-Alive in GET Header

Based on the analysis of this research, six components of GET headers had the potential to be utilized to detect HTTP flooding attacks occurring at the application layer. Table 5 presents the GET header components and the proposed detection.

Table 5: GET Header Components

No	Proposed Detection	GET Header Component
1.	Detects existence of GET headers for each request received.	HTTP user agent HTTP accept language HTTP connection HTTP accept encoding
2.	Detects false and irrelevant query in GET request.	HTTP request query
3.	Detects irrelevant URL	HTTP referrer

8. PROPOSED DETECTION ALGORITHMS

In this section, the process flows to detect HTTP DDoS attacks, as well as the components of the proposed detection system, are given. The detection component comprises of GET header components, which inspired by past studies [36, 37]. The proposed detection method, called FARGO fast algorithm to recognize HTTP DDoS attacks, consists of three detection algorithms and was designed in pairings where an administrator can choose to utilize GET header inspection, query inspection or HTTP referrer inspection. FARGO was designed to be located at the front-end of the web server and behind a firewall. This design is also known as a victim-end defense. [15] explained a victim-end defense as detection placed within a victim border at the front-end of a web server. Figure 9.1 presents the logical architecture for this detection strategy. A detailed explanation of the detection process for each detection algorithm is set out in Sections 8.2 to 8.3. FARGO was developed using VB.Net programming language by adapting GET header components. Figure 3 illustrated the architecture the position of the detection algorithm.

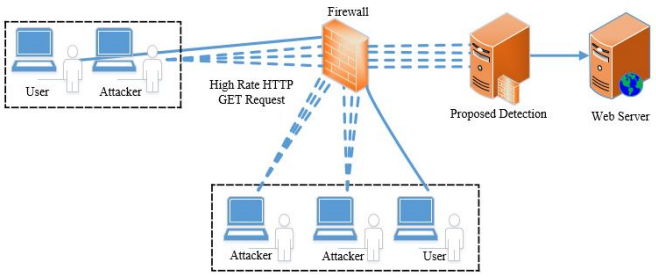

Figure 3: Logical Architecture of Proposed Detection Method

8.1 GET HEADER INSPECTION ALGORITHM

Each request received required adherence to GET header rules. This inspection was performed by the GET header inspection algorithm. The GET request connection had to present common GET headers with the same genuine request. A GET request that successfully passed this inspection was also required to pass a sub-inspection. The sub-inspection examined the connection status of either “Keep-Alive”, “keep-alive” or “close”. If the received GET header had a status of “close”, the connection was detected as an HTTP DDoS attack. The use of upper case or lower-case letters for “Keep-Alive” depends on the web browser. Internet Explorer and Google Chrome use “Keep-Alive”, while Mozilla Firefox uses “keep-alive”. All the components inspected by this algorithm had been tested as components used by authentic users to browse the content of a web server. Figure 4 presents the algorithm.

Algorithm 1—HTTP GET Header Inspection

```
1.  Check GETHeader Similarities
2.  If GET_Header Contain(GETHeader Similarities equal to normal)then
3.  If Header_Connection Status = "Keep-Alive" or "keep-alive" then
4.  Call CleanTraffic
5.  Else
6.  Call DetectHTTP_DDoS
7.  End If
8.  End If
9.  If Header_Connection ="close" then
10. Call DetectHTTP_DDoS
11. End If
12. End if
13. Sub CleanTraffic()
14. Capture IP_Address and Valid GET_Headers
15. Forward traffic to web server
16. End Sub
17. Sub DetectHTTP_DDoS()
18. Detect HTTP DDOS
19. End Sub
```

Figure 4: GET Header Inspection Algorithm

8.2 HTTP REQUEST QUERY INSPECTION ALGORITHM

The use of a query in HTTP GET requests makes a request much closer to a human request. Furthermore, with the help of automated tools, the query can be simply generated. Hence, a query inspection algorithm was introduced to recognize bogus queries generated by HTTP DDoS attacks. A received GET request was inspected to assess whether it contained a request query or not. A GET request had to contain a query with an upper-case letter, and special characters were rejected. This strategy was based on user’s behavior when querying

information on a web server where it was noted that users seldom included upper case letters or other characters. They used lower case letters such as “a”, “b”, or “c” to “z”. The query also had to be short and not longer than four letters. DDoS attacks try to mimic human language and generate words that can be understood by humans, but that situation was not relevant here as the algorithm used compared the query received with a string database. The string database contained a list of keywords that related to a web server. A GET request was marked as genuine by this algorithm if it passed all inspections or did not contain a query in the GET request. Figure 5 displays the algorithm.

Algorithm 2 – HTTP Request Query

```
1.  Check HTTP Query in HTTP GET Request
2.  If GET_Request.Query = false then
3.  Call CleanTraffic
4.  ElseIf CaptureQuery.Contain (Malicious Character) then
5.  Call DetectHTTP_DDoS
6.  ElseIf CaptureQuery.Contain <> Query_Database
7.  Call DetectHTTP_DDoS
8.  Else
9.  Call CleanTraffic
10. End If
11. Sub CleanTraffic ()
12. Capture IP_Address and Valid GET_Headers
13. Forward traffic to web server
14. End Sub
15. Sub DetectHTTP_DDoS()
16. Detect HTTP DDOS
17. End Sub
```

Figure 5: Request Query Inspection Algorithm

8.3 HTTP REFERRER INSPECTION ALGORITHM

HTTP DDoS attacks have the ability to mimic human access. Hence, an attacker makes use of HTTP referrals to make a GET request more genuine and appear to come from a human access pattern. The HTTP referrer inspection algorithm scrutinized HTTP referrals in each GET header to inspect their relevancy. For example, a university website should have links related to the website such as from search engines like Google or any related education site. If the referral contains a link from an online shopping website, the connection is suspicious. When a value of HTTP referral is referred to the URL, it shows the previous website address of the current website being accessed [38]. Figure 6 presents the algorithm.

Algorithm 3 – HTTP Referrer Inspection

```
1.  Check HTTPReferrer In GETHeader
2.  If HTTPReferrer Inside WhiteList.txt
3.  Call CleanTraffic
4.  ElseIf HTTPReferrer Inside BlackList.txt
5.  Call DetectHTTP_DDoS
6.  end if
7.  if HTTPReferrer equal to denied WebCategory
8.  Call DetectHTTP_DDoS
9.  Else
10. Call CleanTraffic
11. end if
12. Sub CleanTraffic
13. Capture IP_Address and Valid GET_Headers
14. Forward traffic to web server
15. End Sub
16. Sub DetectHTTP_DDoS
17. Detect HTTP DDOS
18. Capture IP_Address and Invalid GET_Headers
19. Insert Into BlackLists_Database (HTTP_Referral)
20. End Sub
```

Figure 6: HTTP Referrer Inspection Algorithm

9. EVALUATION ARCHITECTURE

This section describes the experiments conducted and the performance metrics used to evaluate the proposed approach. All proposed algorithms were tested before being compared. Test results showed that DDoS attacks could be fully detected by using the proposed algorithms. They also showed that utilizing GET header components and strategies to recognize the attacks worked as expected. However, a comparison with past studies showed a detection performance drop. The researchers believe that the performance drop was due to the huge number of requests generated by the DDoS attacks, which made the detection device unable to handle such requests. The comparison utilized real HTTP DDoS tools such as HULK, GoldenEye and HOIC, which were also used by prior studies to evaluate their proposed detection methods. Genuine HTTP GET requests and HTTP DDoS attacks were executed simultaneously. The measurement unit to evaluate detection performance utilized a confusion matrix to observe true positives and false positives. Figure 7 shows the physical detection architecture for evaluation, while Table 6 presents the confusion matrix.

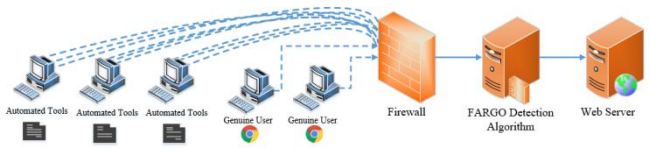

Figure 7: Experiment Architecture

Table 6: Confusion Matrix

Detection Matrix	Description
True Positive (TP)	Number of attacks correctly classified as attacks
False Positive (FP)	Number of normal traffic incorrectly recognized as attacks
Confusion Matrix Calculation Formula	
True Positive Rate	TP / (TP + FN)
False Positive Rate	FP / (FP + TN)

10. RESULTS AND DISCUSSION

The detection performance shown by the FARGO fast algorithm method displayed expected performance where all detection matrices provided 100% detection for true positive, true negative, precision, false positive, false negative and accuracy. The results showed that inspection of GET headers to recognize HTTP DDoS attacks in real time was successfully achieved. The GET header inspection algorithm ensured that genuine components must appear during HTTP transactions and failure to present the components would result in detection of an HTTP DDoS attack. The results also showed that the first algorithm had the capability to differentiate between complete and incomplete GET header requests performed either by an attacker or an authentic user. Figure 8 shows the performance graph for Algorithm 1.

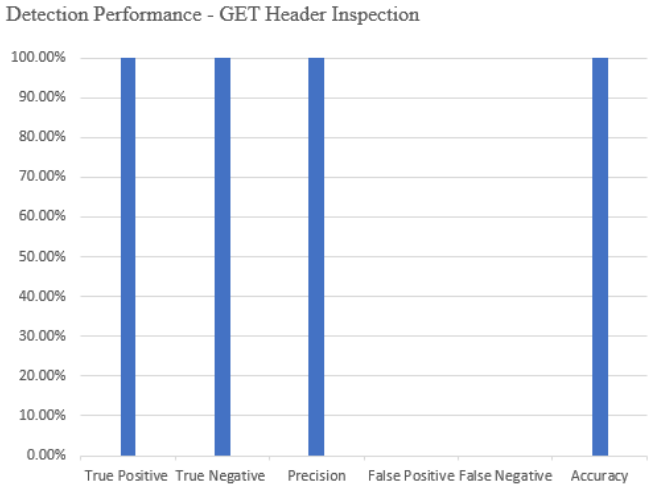

Figure 8: Detection Performance of GET Header Inspection

During HTTP GET request transaction a web server will be received GET header from client to inform the identity of the source request. However, during the occurrence of HTTP DDoS, the pattern is absolutely differed, which make FARGO reach 100% detection for matrix measured. Although HTTP DDoS had an ability to manipulate GET header components to look genuine, part of the components not able to be manipulated by the attacker such as Accept-Language. Gou, Bai [35] explained an authentic GET request deliver complete components. Figure 9 illustrates genuine GET header component.

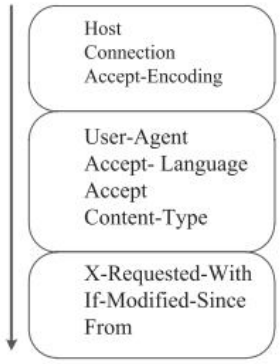

Figure 9: Genuine GET Header Component

The outcomes from the request query inspection algorithm, when checking against GET header requests, either contained a request query or were empty. The outputs from this algorithm signified that false GET header requests were able to be distinguished satisfactorily. The outputs also provided clear indicators to recognize bogus queries sent by HTTP DDoS attacks to mimic human requests. The achievement of these results also showed that extraction of request queries from received GET headers worked smoothly in real time. Figure 10 shows the performance graph for Algorithm 2.

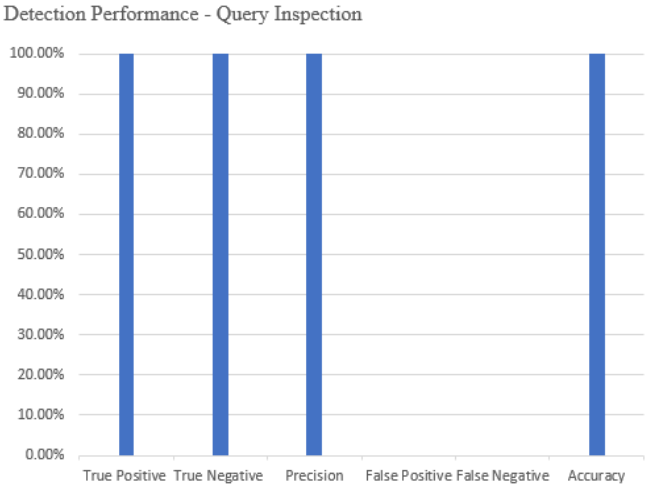

Figure 10: Detection Performance - Query Inspection

Results at section 7.0 indicate HTTP DDoS delivery unreadable format for request query in HTTP GET request. Human query is readable, which contradict with the query patterns deliver by HTTP DDoS. HTTP DDoS generate false query to emulate authentic request to conceal their activity to make a web server overwhelmed with plenty of requests. Furthermore, there is no mechanism to detect HTTP DDoS through request query. FARGO performs inspection against the request query to determine the relevancy of query in GET request, which makes the detection matrix reach 100% detection. A web server adopts in this research hosted a web

page that has been design not to accept query. However, results received at section 8.0 indicate false query received by a web server which a sign of HTTP DDoS has been occurred. The performance of the HTTP referrer detection algorithm also indicated a positive output. Results gained from this algorithm showed that irrelevant HTTP referrers were able to be recognized as expected. Figure 11 shows the performance graph for Algorithm 3.

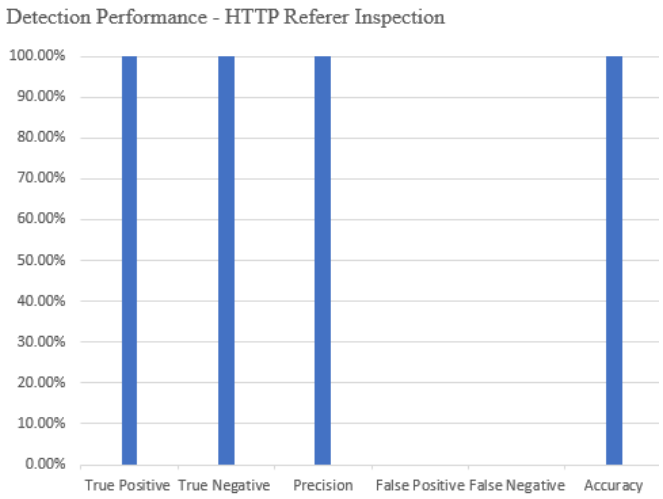

Figure 11: Detection Performance - HTTP Referrer Inspection

HTTP referrer refer to past web site address accessible by users to browse the current web page. Occurrence of HTTP DDoS attack shows at section 8.0 the attack capable to mimic valid URL of past web page. This indicates detection of HTTP DDoS become difficult due to its ability to mimic authentic GET request as the GET header component is appeared with correct value assign to GET header components. 100% detection gained by FARGO due to inspection against HTTP referral value to check either the value of HTTP referrer is relevant to be referral of the current page accessed. Real attack was launched against test web site as explained at section 4.0 and existence of not related URL as referrer proof that HTTP DDoS utilize HTTP referrer in HTTP GET request to make a request look genuine. Examine the existence of the component will not work as the component is not consistently presented during GET request. The header is not presented when refer to local request [39].

11. COMPARISON WITH PAST STUDIES

Comparison with past studies to indicate an improvement makes by proposed detection named as FARGO to detect HTTP DDoS. Comparison was done by using actual HTTP DDoS attack script which same used by prior studies.

11.1 COMPARISON WITH TIMES SERIES PREDICTION MODEL

Comparison with past studies disclosed that the proposed detection method was still able to maintain its performance under certain specified circumstances. A comparison was carried out against the Times Series Prediction Model of [40], utilizing the HOIC DDoS attack tool (which was also used by that study) to evaluate their proposed detection method. The FARGO detection algorithms showed an improved result of 11.30% for the true positive rate and an 8.90% enhancement for the false positive rate. The HOIC is a real HTTP DDoS attack tool and delivers a minimum number of GET requests. Due to this type of delivery, the proposed algorithm was able to distinguish whether the requests came from a DDoS attack or from authentic traffic.

The FARGO detection algorithms also performed inspections of common GET headers delivered by a requestor to a web server. Incomplete information during a transaction of a GET request resulted in detection of an HTTP DDoS attack, which meant that the FARGO detection algorithms could fully recognize the attack. The achievement of this result showed that DDoS attacks at the application layer could be detected with high true positives if the attack provides minimal information in the GET request, as genuine traffic will provide more information during the transaction between client and web server. Figure 12 indicate the detection performance.

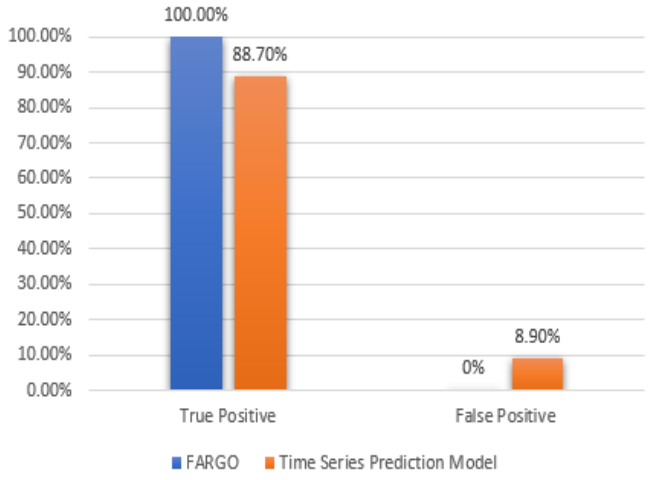

Figure 12: Comparison with the Time Series Prediction Model

11.2 COMPARISON WITH LOGISTIC REGRESSION ANALYSIS OF INFORMATION ENTROPY

A final comparison using logistic regression analysis of information entropy produced expected results with improvements of 1.16% for true positive rates and 1.70% for false positive rates. The true positive rate showed that the proposed detection method correctly recognized bogus traffic,

which caused the proposed detection method to acquire 99.95%. The false positive rate indicated no missed classifications for authentic traffic detected as false traffic, which led to a result of 0.00% for the false positive rate.

Nevertheless, there were instances of attack traffic being recognized as genuine, which caused the detection rate for true positives to drop. The performance drop was due to several possible reasons, such as missed detection due to the high amount of GET requests generated by the attack and traffic pattern similarities with genuine requests, which made the proposed detection method unable to differentiate between authentic and forged requests. Table 7 presents the comparison results while Figure 13 provide graphical views of the detection performance.

Table 7: Comparison with Past Studies

No	Detection Technique	True Positive Rate	False Positive Rate
1.	Times Series Prediction Model [40]	88.70%	8.90%
2.	FARGO (Proposed Detection Method)	100.00%	0.00%
3.	**Improvement**	11.30%	8.90
1.	Logistic Regression [37]	98.79%	1.70%
2.	FARGO (Proposed Detection Method)	99.95%	0.00%
3.	**Improvement**	1.16%	1.70%

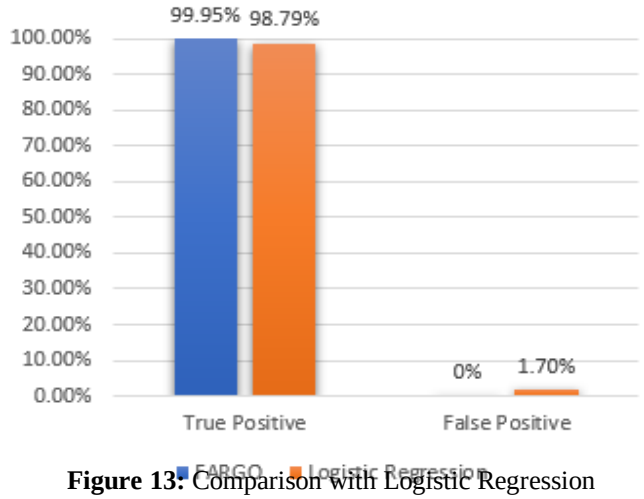

Figure 13: Comparison with Logistic Regression

A result at section 11.1 and comparison with Times Series Prediction Model shows the proposed detection able to maintain it performance. However, comparison with Logistic Regression detection technique indicates a performance drop due to several reasons such as: This research adopts real hardware with actual attack scripts to perform experiment and dealing with real atmosphere of HTTP DDoS. Hence limitation against devices is one of the main factors the performance was dropped. Each device had a specific workload can handle. Furthermore, there are no such devices can handle the infinity workload. Higher speed GET request generate by HTTP DDoS is the second reason why the detection performance drop. Besides that, the higher speed is continuously generating which make the workload of the detection device gradually increase until overwhelmed by massive GET request. Apart from that the proposed detection requires time to scrutinize each received GET request to determine the source request is authentic of comes from malicious.

Appearance of HTTP version 2 to provide enhancement against HTTP version 1.1. However, HTTP 2 still utilize the same GET header components utilize by HTTP 1.1. Header compression introduce by HTTP 2 reduce number of bytes sending by each request. Ludin and Garza [41] explained only unique byte are sending to a web server unlike HTTP 1.1 the entire bytes will be resent. Table 8 illustrate the explanation where second request only sent the unique byte, in this case the unique byte is only 10 bytes.

Table 8: HTTP 2 GET headers

Request 1 # Total Bytes 220	Request 2 # Total Byte 230
: Authority: www.akamai.com : Method: GET : Path:/ : scheme: https accept: text/html,application/xhtml+xml accept-anguange:en-US,en;q=0.8 cookie:last_page=286A7F3DE upgrade-insecure-request:1 user-agent:mozilla http2	: Authority: www.akamai.com : Method: GET : Path:/**style.css** : scheme: https accept: text/html,application/xhtml+xml accept-languange:en-US,en;q=0.8 cookie:last_page=***39 8AB8E8F** upgrade-insecure-request:1 user-agent:mozilla http2

* **The bold is the different**

12. CONCLUSION

All proposed algorithms showed their ability to detect false GET requests instituted by HTTP DDoS attacks and the achievement of this target illustrated that the detection strategy of utilizing GET header components to perform HTTP DDoS attack detection worked as expected. Real experiments were conducted to indicate that the proposed detection method had the potential to be used in production network environments and revealed that the proposed method worked not only to reach academic targets but also to provide supplementary information in the detection of such attacks.

To ensure that the proposed detection method has the ability to work in a production network environment, a variety of DDoS attack strategies should be further tested. Detection times and the workloads that can be supported by the proposed detection method are components that need to be tested in the future. HTTP DDoS attacks launched through a proxy and behind NAT should be another research area. In addition, the use of other platforms to execute HTTP DDoS attacks, such as IoT devices, should be examined. Further, the proposed method could have the potential to detect DDoS attacks classified as low rate and flash crowd with enhancements of several sections in the proposed detection algorithms. The proposed detection is possible to work with HTTP 2 as GET the header components to form detection in this research is still utilized by HTTP 2. However, DDoS attack occurs at the application layer employ HTTP 2 are still uncertain. Moreover, HTTP 1.1 is still widely used, and require time to entirely move to HTTP 2.

ACKNOWLEDGMENT

This research paper was compiled at University Technology Malaysia (UTM).

REFERENCES

1. Zolotukhin, M., et al. *Increasing web service availability by detecting application-layer DDoS attacks in encrypted traffic*. in *Telecommunications (ICT), 2016 23rd International Conference on (pp. 1-6). IEEE*. 2016. IEEE. https://doi.org/10.1109/ICT.2016.7500408
2. Beitollahi, H. and G. Deconinck, *Analyzing well-known countermeasures against distributed denial of service attacks.* Computer Communications, 2012. **35**(11): p. 1312-1332. https://doi.org/10.1016/j.comcom.2012.04.008
3. Subramanian, K., P. Gunasekaran, and M. Selvaraj, *Two Layer Defending Mechanism against DDoS Attacks.* International Arab Journal of Information Technology (IAJIT), 2015. **12**(4).
4. Yuan, X., C. Li, and X. Li. *DeepDefense: Identifying DDoS Attack via Deep Learning*. in *Smart Computing (SMARTCOMP), 2017 IEEE International Conference on*. 2017. IEEE. https://doi.org/10.1109/SMARTCOMP.2017.7946998
5. Singh, K., P. Singh, and K. Kumar, *Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges.* Computers & Security, 2017. **65**: p. 344-372. https://doi.org/10.1016/j.cose.2016.10.005
6. Ni, T., et al., *Real-Time Detection of Application-Layer DDoS Attack Using Time Series Analysis.* Journal of Control Science and Engineering, 2013. **2013**: p. 1-6. https://doi.org/10.1155/2013/821315
7. Ziyad R. Al Ashhab, et al., *Detection of HTTP Flooding DDoS Attack using Hadoop with MapReduce : A Survey*. International Journal of Advanced Trends in Computer Science and Engineering, 2019. **Volume 8, No.1, January – February 2019**: p. 1-7.
8. Aamir, M. and S.M.A. Zaidi, *Clustering based semi-supervised machine learning for DDoS attack classification.* Journal of King Saud University - Computer and Information Sciences, 2019. https://doi.org/10.1016/j.jksuci.2019.02.003
9. Kolias, C., et al., *DDoS in the IoT: Mirai and Other Botnets.* Computer, 2017. **50**(7): p. 80-84. https://doi.org/10.1109/MC.2017.201
10. Amin Salih Mohammed, et al., *Analysis of Mobile IP Wireless Networks in 5G.* International Journal of Advanced Trends in Computer Science and Engineering, 2019. **Volume 8, No.1.2, 2019**: p. 1-4.
11. Hameed, S. and U. Ali, *HADEC: Hadoop-based live DDoS detection framework.* EURASIP Journal on Information Security, 2018. **2018**(1): p. 11. https://doi.org/10.1186/s13635-018-0081-z
12. Cheng, J., et al., *Adaptive DDoS Attack Detection Method Based on Multiple-Kernel Learning.* Security and Communication Networks, 2018. **2018**: p. 1-19. https://doi.org/10.1155/2018/5198685
13. Rahman, R.u., D.S. Tomar, and J. A.V, *Application Layer DDOS Attack Detection Using Hybrid Machine Learning Approach.* International Journal of Security and Its Applications, 2017. **11**(4): p. 85-96. https://doi.org/10.14257/ijsia.2017.11.4.07
14. Beitollahi, H. and G. Deconinck, *ConnectionScore: a statistical technique to resist application-layer DDoS attacks.* Journal of Ambient Intelligence and Humanized Computing, 2013. **5**(3): p. 425-442. https://doi.org/10.1007/s12652-013-0196-5
15. Hoque, N., D.K. Bhattacharyya, and J.K. Kalita, *Botnet in DDoS Attacks: Trends and Challenges.* IEEE Communications Surveys & Tutorials, 2015. **17**(4): p. 2242-2270.
16. Zargar, S.T., J. Joshi, and D. Tipper, *(2013-Q1) A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks.* IEEE communications surveys & tutorials, 2013. **15**(4): p. 2046-2069.
17. Ranjan, S., et al., *DDoS-Shield: DDoS-Resilient Scheduling to Counter Application Layer Attacks.*

IEEE/ACM Transactions on Networking, 2009. **17**(1): p. 26-39.
18. Yadav, S. and S. Subramanian. *Detection of Application Layer DDoS attack by feature learning using Stacked AutoEncoder*. in *Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on*. 2016. IEEE.
19. Kumar, V. and K. Kumar. *Classification of DDoS attack tools and its handling techniques and strategy at application layer*. in *Advances in Computing, Communication, & Automation (ICACCA)(Fall), International Conference on (pp. 1-6)*. 2016. IEEE.
20. Prasad, K.M., A.R.M. Reddy, and K.V. Rao, *BARTD: Bio-inspired anomaly based real time detection of under rated App-DDoS attack on web*. Journal of King Saud University - Computer and Information Sciences, 2017.
21. Rai, A. and R.K. Challa, *Survey on Recent DDoS Mitigation Techniques and Comparative Analysis.* 2016: p. 96-101. https://doi.org/10.1109/CICT.2016.27
22. Idhammad, M., K. Afdel, and M. Belouch, *Detection System of HTTP DDoS Attacks in a Cloud Environment Based on Information Theoretic Entropy and Random Forest.* Security and Communication Networks, 2018. **2018**: p. 1-13.
23. Behal, S., K. Kumar, and M. Sachdeva, *D-FAC: A novel ϕ -Divergence based distributed DDoS defense system.* Journal of King Saud University - Computer and Information Sciences, 2018. https://doi.org/10.1016/j.jksuci.2018.03.005
24. Singh, K., P. Singh, and K. Kumar, *User behavior analytics-based classification of application layer HTTP-GET flood attacks.* Journal of Network and Computer Applications, 2018. **112**: p. 97-114.
25. Zhao, Y., et al., *A Classification Detection Algorithm Based on Joint Entropy Vector against Application-Layer DDoS Attack.* Security and Communication Networks, 2018. **2018**: p. 1-8.
26. Sreeram, I. and V.P.K. Vuppala, *HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm.* Applied Computing and Informatics, 2017.
27. Aborujilah, A. and S. Musa, *Cloud-Based DDoS HTTP Attack Detection Using Covariance Matrix Approach.* Journal of Computer Networks and Communications, 2017. **2017**: p. 1-8.
28. Singh, K.J. and T. De, *MLP-GA based algorithm to detect application layer DDoS attack.* Journal of Information Security and Applications, 2017. **36**: p. 145-153.
29. Hoque, N., H. Kashyap, and D. Bhattacharyya, *Real-time DDoS attack detection using FPGA.* Computer Communications, 2017. **110**: p. 48-58.
30. Liao, Q., et al., *Application layer DDoS attack detection using cluster with label based on sparse vector decomposition and rhythm matching.* Security and Communication Networks, 2015. **8**(17): p. 3111-3120.
31. Jazi, H.H., et al., *Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling.* Computer Networks, 2017. **121**: p. 25-36. https://doi.org/10.1016/j.comnet.2017.03.018
32. Behal, S. and K. Kumar, *Trends in Validation of DDoS Research*, in *International Conference on Computational Modeling and Security (CMS 2016)*. 2016, Elsevier. p. 7-15.
33. Jaafar, G.A., S.M. Abdullah, and S. Ismail, *Review of Recent Detection Methods for HTTP DDoS Attack.* Journal of Computer Networks and Communications, 2019. **2019**: p. 1-10.
34. Behal, S. and K. Kumar, *Detection of DDoS attacks and flash events using novel information theory metrics.* Computer Networks, 2017. **116**: p. 96-110. https://doi.org/10.1016/j.comnet.2017.02.015
35. Gou, G., et al., *Discovering abnormal behaviors via HTTP header fields measurement.* Concurrency and Computation: Practice and Experience, 2017. **29**(20): p. e3926.
36. Saleh, M.A. and A. Abdul Manaf, *A Novel Protective Framework for Defeating HTTP-Based Denial of Service and Distributed Denial of Service Attacks.* ScientificWorldJournal, 2015. **2015**: p. 238230.
37. Yadav, S. and S. Selvakumar. *Detection of application layer DDoS attack by modeling user behavior using logistic regression*. in *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2015 4th International Conference on*. 2015. IEEE.
38. Reid, F., *4 - HTTP: Communicating with Web Servers*, in *Network programming in .NET*, F. Reid, Editor. 2004, Digital Press: Burlington. p. 87-130.
39. Fielding, R. and J. Reschke, *Hypertext transfer protocol (HTTP/1.1): Semantics and content.* 2014. https://doi.org/10.17487/rfc7231
40. Wang, Y., et al. *A novel approach for countering application layer DDoS attacks.* in *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2017. https://doi.org/10.1109/IAEAC.2017.8054326
41. Ludin, S. and J. Garza, *Learning HTTP/2: A Practical Guide for Beginners*. 2017: " O'Reilly Media, Inc.".