sensors

Article

Recent Analysis of Forged Request Headers Constituted by HTTP DDoS

Abdul Ghafar Jaafar *, Saiful Adli Ismail, Mohd Shahidan Abdullah, Nazri Kama, Azri Azmi and Othman Mohd Yusop

Razak Faculty of Technology and Informatics, Universiti Teknologi Malaysia (UTM), Kuala Lumpur 54100, Malaysia; saifuladli@utm.my (S.A.I.); mshahidan@utm.my (M.S.A.); mdnazri@utm.my (N.K.); azriazmi@utm.my (A.A.); othmanyusop@utm.my (O.M.Y.)

* Correspondence: afastars@gmail.com

Received: 18 May 2020; Accepted: 23 June 2020; Published: 8 July 2020

Abstract: Application Layer Distributed Denial of Service (DDoS) attacks are very challenging to detect. The shortfall at the application layer allows formation of HTTP DDoS as the request headers are not compulsory to be attached in an HTTP request. Furthermore, the header is editable, thus providing an attacker with the advantage to execute HTTP DDoS as it contains almost similar request header that can emulate a genuine client request. To the best of the authors' knowledge, there are no recent studies that provide forged request headers pattern with the execution of the current HTTP DDoS attack scripts. Besides that, the current dataset for HTTP DDoS is not publicly available which leads to complexity for researchers to disclose false headers, causing them to rely on old dataset rather than more current attack patterns. Hence, this study conducted an analysis to disclose forged request headers patterns created by HTTP DDoS. The results of this study successfully disclose eight forged request headers patterns constituted by HTTP DDoS. The analysis was executed by using actual machines and eight real attack scripts which are capable of overwhelming a web server in a minimal duration. The request headers patterns were explained supported by a critical analysis to provide the outcome of this paper.

Keywords: DDoS; HTTP DDoS; GET Headers

1. Introduction

Due to the increase of HTTP DDoS attacks against web servers, the need for a dataset for the attack is pertinent. However, recent datasets for HTTP DDoS attacks are not publicly available due to several reasons as noted by many studies. Behal and Kumar [1] conducted a study on DDoS dataset utilized by prior researchers and explained that publicly available dataset had its limitations as most of the datasets were captured from the network layer thus obscuring information that are available at the application layer. The lack of current dataset disclosing recent false GET request headers produced by HTTP DDoS attacks provides little knowledge on the pattern types and ways they are executed. HTTP DDoS utilizes request headers as a dummy to conceal malicious activity and send substantial amount of HTTP requests to overwhelm a web server. Aside from that, prior studies stated minimal request headers pattern constituted by HTTP DDoS and unable to deliver detailed explanation. The experiment which specifically points to forged request headers generated by HTTP DDoS with usage of a recent attack script is still not presented in research community. Hence, a comprehensive experiment is required to deal with real devices and attack scripts to disclose the current pattern of forged request headers adopted by attackers to make the attack appear authentic.

The dependency on publicly available dataset when conducting experiments for HTTP DDoS should be avoided as the datasets contain old attack patterns which contradict with the current attack patterns produced by current attack scripts. Jazi, et al. [2] explained that the lack of dataset poses a major difficulty to conduct an experiment. Singh, et al. [3] lamented that many studies employed old datasets and used university or organization's web logs as the benchmark to evaluate their work and noted that comparing their work with an old dataset is meaningless. Thus, generating dataset that is close to the actual environment using network topologies is required [4]. Usage of current DDoS dataset permit researchers to acquire the latest atmosphere the forged request header employed by cyber intruders to make the attack traffic look authentic. The obsolete dataset like (1) KDD Cup Dataset 1999, (2) CAIDA DDoS Attack Dataset 2007, (3) Environmental Protection Agency (EPA) HTTP dataset 1995, (4) DARPA DDoS attack dataset 2009, (5) Clarknet 1995, (6) NASA 1995 has been noted by prior studies [1,5,6].

This paper reveals the forged request headers constituted by HTTP DDoS starting with Section 5 which discusses HTTP manipulation while Section 6 discloses invalid request headers utilized by attackers when launching HTTP DDoS attacks. Both sections help future researchers to determine the appropriate detection that can be applied when detecting malicious request produced by HTTP DDoS. Detection is the first defense to recognize malicious traffic unlike other methods such as prevention, mitigation and monitoring. Munivara Prasad, et al. [7] noted that detection is the first step to be executed followed by mitigation. This is because clean and attack traffics can be differentiated at the detection phase. According to Zargar, et al. [8], the deployment of the detection stage can be done either at the source, intermediate network, destination or a combination of them. HTTP DDoS generates massive HTTP requests to overwhelm a web server. Thus, selecting the most appropriate detection technique is very crucial to avoid higher rate of false positive and false negative. To ensure that a higher rate of true positive and true negative can be achieved, three detection methods broadly known as signature-based, anomaly-based, and hybrid must be deployed with precaution. This is to ensure that any detection flows that have been designed will be compatible with any of the intrusion prevention systems [9]. Thorough explanation pertaining to the different types of detection methods are presented in the following sub-sections.

1.1. Signature-Based Detection

Osanaiye, et al. [10] elaborated that a set of policies were applied to inspect incoming traffics and to utilize the knowledge database to store known attack patterns. The database was constructed by collecting all related attack patterns [11]. Traffics receive were compared with knowledge database which therefore required the database to be frequently updated to ensure the attack traffic can be detected [10–12]. However, adoption of absolute database leads to a misclassification of false positive and false negative due to a high possibility of occurrence. Cheng, et al. [13] noted that this approach requires information from the available dataset to capture the behavior and information about various attacks. The signature-based detection was built based on the network behavior and there are also various terms that have been used to describe it including misuse, knowledge-based, rule-based and pattern-based detection [13,14]. The signature-based detection depends on a knowledge-based database. A set of policies is required to detect malicious traffics; hence, usage of appropriate policy is highly recommended to be applied in order to recognize any attacks. An inaccurate policy will increase the occurrence of DDoS traffic to reach a web server. Aside from that, the signature-database requires frequent updates which will increase the storage capacity. As explained by Cheng, et al. [13], this approach uses available dataset to gain information about the attack. However, not all dataset is shared publicly due to security reasons. Due to this, collaboration with other parties is highly required by sharing the attack pattern to ensure that the database is constantly updated. Usage of security tools by performing experiments can also predict any incoming attack, which will increase the maturity of the knowledge database.

1.2. Anomaly-Based Detection

Myint Oo, et al. [14] noted that this detection type utilized a machine learning concept and is also known as an outlier and performance-based detection [13]. Cheng, et al. [13] further elaborate that anomaly-based detection is capable of detecting new and unknown attack types. Osanaiye, et al. [10] explained that anomaly-based detection utilizes a traffic's profile of typical behavior to detect malicious behavior [10,12]. The traffic profile must operate for several days to acquire normal behavior access pattern. This technique utilizes two approaches noted as training and detection phases. The training phase relied on the input to obtain regular access behavior. The type of detection consisted of several categories such as supervised, semi supervised and unsupervised. In supervised learning, all data are labeled to presume the results while in a semi supervised learning, the data are not entirely labeled, and in unsupervised data, all data are unlabeled as it will learn based on the input data. Anomaly-based detection requires several days to thoroughly learn about normal user access behavior. Adopting this approach is time-consuming as all traffic behaviors must be stored in the traffic profile. An inaccurate classification will occur if the training profile is used before obtaining the entire behavior. The traffic profile also requires constant updates, especially for new access pattern, which is also equal to the signature-based detection as failure to update will result in malicious traffic to reach a web server.

1.3. Hybrid-Based Detection

This detection method merged signature-based and anomaly-based techniques to form a hybrid detection system [10,13,15]. Hybrid-based detection utilizes a feature from both techniques to obtain a higher detection rate [10]. Cheng, et al. [13] highlight that hybrid detection can help analyst to detect both normal and malicious system behavior to increase the capabilities for monitoring the detection system.

2. Contribution of the Paper

This paper aims at giving the reader thorough insights into the forged request header patterns generated by HTTP DDoS. To the best of our knowledge, there are no current studies conducted that look into HTTP DDoS with the adoption of recent attack scripts to disclose forged request headers adopted by the attack. The request headers pattern for HTTP DDoS categorized as slow rate HTTP DDoS has been disclosed by several studies [16–18]. However, the forged request headers for HTTP DDoS categorized as session flooding and request flooding are not excessively revealed. Hence this paper is an extension of surface knowledge provided by prior studies and contributes as follows:

- Eight forged request header patterns produced by HTTP DDoS have been revealed through extensive analysis by adopting actual HTTP DDoS attack scripts. Critical analysis has been made to acquire similarities and differences and a summary of the request header patterns.
- The HTTP DDoS architecture has been introduced based on the usage of eight actual attack scripts. The attack scripts have been adapted to have different attack strategies which are capable of executing HTTP DDoS through Direct Attack, Proxy, and Spoof IP address.
- A taxonomy of HTTP DDoS attack patterns has been created by incorporating forged request headers noted by prior studies with false request headers unveiled in this study.
- A recent attack script to execute HTTP DDoS has been assigned to the existing attack strategy suggested by prior studies.

3. DDoS Attack at the Application Layer

The protocol known as Transmission Control Protocol (TCP) and HTTP utilized by attackers to gain server resources lead to the inability of a web server to handle client requests [19]. Hoque, et al. [20] mentioned that the difficulties to detect DDoS at the application layer is due to three factors. The first factor is due to obscurity as an HTTP protocol uses TCP and UDP connections to run its operation thus leading to complications to differentiate between legitimate and illegitimate traffic. The second factor is efficiency as DDoS attack at the application layer only requires fewer connections to initiate an attack. The third and final factor is lethality as the attack has the capability to overwhelm a web server instantaneously resulting in service breakdown regardless of the type of hardware and its performance. A DDoS attack which occurs at this layer consumes resources, including CPU and memory [21,22].

4. DDoS Attack Strategy at Application Layer

The existence of DDoS attack strategy executed at the application layer is due to the architecture of a web server. For instance, a web server that handles a website allows users to access a web page without going through the main page. This approach provides convenience for users to search for information without accessing the first page of the website. As a result, attackers are able to launch attacks at different pages of the website. The attack strategy, as described above, indicates that the structure of the website allows the attacker to create several attack strategies to overwhelm a web server through a full HTTP request.

Conversely, for a web-based application that is commonly categorized as an online system, users will not be allowed to access the subpage without passing through the login page, which deny the attacker to execute DDoS at application a different page as explained above. The login page's existence will deny attackers from targeting specific pages as the login page has blocked it; however, having the login page is only suitable for an online system and is inappropriate to be adopted on a website. A website is utilized by many entities to promote and provide info about a specific operation. Execution of DDoS attacks at the application layer employ several attack strategies. This is to ensure that the attack can bypass security devices, and the targeted web server is collapsed and unable to serve a client's request. DDoS attacks are executed at the network layer, which can be easily detected [23]. Due to this, the attacker opts to execute the attack at the application layer to make a web server turn to offline mode therefore incapable of responding to a client's request [3]. The attack strategy to execute DDoS attack at the application layer has been revealed by the same authors [3,6] in different studies as elaborate as follows:

- **Server Load:** Attacker uses botnet to continuously send malicious request against a web server aggressively, which will lead to the server to drop legitimate request as the resources of the web servers are running out.
- **Increasing:** Attacker uses low request value to initiate an attack and slowly increasing the value. This behavior of attack is difficult to detect as malicious HTTP traffic is not sent aggressively to a victim's server during the occurrence of the attack.
- **Constant:** To perform this attack strategy, cyber intruders must specify a specific number of request rate to be sent to the victim's HTTP web server. The request number is referred to as constant, which will be the same as when the botnet sends a malicious request to a web server. Commonly, the request rate is between 100, 200 and 300 per second.
- **Single Web Pages Attack:** Attacker uses any single web page that belongs to a website. The botnet is commanded by the attacker, which will continuously send malicious HTTP requests to web server.
- **Main Page Attack:** Cyber intruders specifically focus on the main page of the websites to deny legitimate users from getting any access. To generate an attack, traffic botnet will be used to repeatedly send a malicious request to a web server. The impact of this attack only occurs on the main page of the website, while the subpages of the website are not affected.
- **Dominant Page Attack**: This category refers to a web page with a greater interest for authentic users to access. The attacker then focuses on that particular page to embark HTTP DDoS attack to prevent a legitimate user from browsing the contents of the web pages. This attack only gives an impact on web pages that have a greater interest for users to browse.
- **Multiple Page Attack:** A cyber attacker will initiate the attack at multiple web pages from a website. The purpose of this technique is to avoid detection as the malicious HTTP request will

imitate the human access pattern. For instance, humans will open more than one web page to find information while surfing websites. When the attack occurs, more than one of the existing web pages will not be accessible as the attacker is interested in targeting multiple web pages.

- **Reply Flood Attack:** The botnet command by the attacker sends an HTTP traffic at an inflated rate to gain a resource of the web server to prevent web server from surfing legitimate HTTP request. The attacks work by gaining human access patterns to prohibit the detection system from blocking the malicious request.
- **Random Attack:** All web pages that belong to a website are not accessible regardless of its category.
- **Rare Change Page Attack:** The common structure of a web system will group a page into a specific group to make the page content more structured and user friendly. Since the arrangement of the web page is grouped, an attacker may compromise the group page by commanding botnet to the web page. Hence, the group web page will be the most targeted pages. Consequently, this attack will prevent a user from opening a web page that belongs to a specific group.
- **Frequent Change Attack:** An attacker performs an attack on a web page that belongs to different categories. This attack will rotate and send a malicious request to distinct web page categories. This attack only affects specific categories of the website. When the attack occurs, other web pages can still be accessed as usual.
- **Hot Pages Attack:** Each web-based system will have frequent open pages. Hence, this situation will give an opportunity for cyber intruders to initiate the attack on the most visited pages to prevent legitimate users from accessing as the main objective for a DDoS attack is to avoid users from opening the pages.
- **Web Proxy Attack:** Attackers use a proxy server as a representative to generate attack traffic. The use of a proxy server to generate attack traffic will cause difficulties in detecting the source of the attack. Multiple proxy servers would be used to generate plenty of HTTP requests to overwhelm the web server.

5. HTTP Manipulation

Currently, HTTP is the protocol which is widely utilized by many web servers. Normal browsing involves two methods known as request and response to allow a user to browse through the web server content. Both methods contain headers which can reveal web applications and their infrastructure [24]. The request comprises headers such as "Accept", "Accept-Encoding", "Accept-Language", "Authorization", "Connection", "Content-Length", and "Content-Type" [25]. Tyson, et al. [26] noted that HTTP become vulnerable to be manipulated by intermediate parties once it communicates across a network.

Another request header which commonly existed in a request is known as "referrer" which refers to the previous website accessed by a user to access the current web site. The referrer is one of the headers adopted by the attacker to generate HTTP DDoS, thus making the request look authentic. Mansoori, et al. [27] noted that referrer was the standard header being exploited, which can be modified and spoofed by firewalls and proxies. A content type is another header that appears in a request which indicates the extension of the HTTP Uniform Resource Identifier (URI) header field [28]. However, the content-type is also vulnerable to be tampered as it appears inconsistently in the normal web browsing and can be declared manually [28]. Manually declaring content-type indicates that the header can be spoofed, which provides advantages to the attacker to emulate this header that resembles an authentic request. A user-agent is one of the compulsory headers that appears in a request header as noted by Liu, et al. [29]. A request will be considered as abnormal if the user-agent is not present. Usage of the user-agent is vital as it is used to determine the platform adopted by the user including the type of web browser, version, operating system, etc. More importantly, the information included in the user-agent is utilized to select the appropriate web content view which is either mobile or desktop view. Without the information provided by the user-agent, smartphone users may face difficulties in viewing the web content compared to the desktop view as there are differences between the two types of views.

However, according to La, et al. [30], the user-agent can be modified by the attacker to perform the attack such as through Simple Query Language (SQL) injection, cross-site scripting and DoS.

Although attackers take advantage of the drawbacks that exist in the request header to execute an attack, security researchers can adopt the headers to design a defense system to recognize malicious traffics that used request headers to make the request looks authentic. Niu, et al. [31] propose a malware detection by using request headers such as "URI", "Host", "User-Agent", "Request-Method", "Request-Version", "Accept", "Accept-Encoding", "Connection", "Content-type", "Cache-Control", and "Content-length". They explained that a malware requires to establish connection to Command and Control (C&C) server which has shorter request message. Saleh and Abdul Manaf [32] combined non-HTTP features with "user-agent", "accept" and "host" while Yadav and Selvakumar [33] also adopted non-HTTP features to be combined with request headers known as "User-Agent" and "Referrer" in providing solutions to detect HTTP DDoS attacks. Malicious request traffic has a specific pattern which can be recognized through detection. Hence, Aceto and Pescape [34] proposed a sensor device that is capable of recognizing HTTP manipulation. The sensor utilizes HTTP response code to send message to a requestor once an HTTP has been tampered. Niakanlahiji, et al. [24], on the other hand, adopted the response header in recognizing phishing websites.

Adoption of attributes existed at the application layer such as request and response headers provided significant impacts in detecting of malicious traffics. This has been proven by prior studies, as noted above, that use the application layer headers capable of detecting an attack. HTTP DDoS is complex to detect due to its similarities with a legitimate request. Therefore, to recognize this attack, the current forged headers produced by the attack require to reveal excessively to acquire current header pattern adopted by the attacker in manipulating request headers. Section 5 discloses extensively the request headers constituted by HTTP DDoS.

6. HTTP DDoS Request Headers

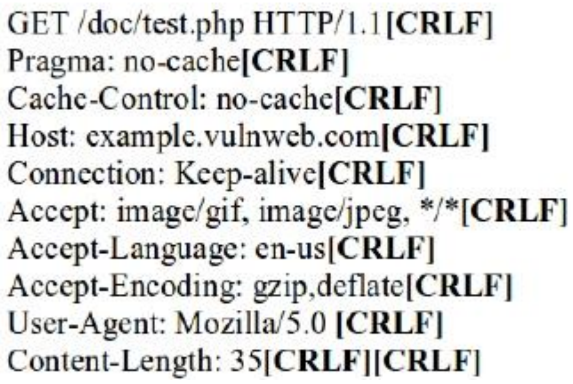
DDoS at the application layer adopted request headers to make an HTTP request looks authentic. Consequently, the attack is complex to be recognized. There are several past studies that reveal the request headers employed by DDoS attacks and are further explained in the following sections.

6.1. HTTP DDoS Request Headers (Session Flooding and Request Flooding)

The request headers adopted by HTTP DDoS are categorized as session flooding and request flooding which are capable of mimicking a genuine user request [35–38]. Aside from that, the attack has the same syntax and is delivered via multiple HTTP requests in different HTTP formats [35]. Sreeram and Vuppala [37] contend that an excessive search request and login is one of the patterns of HTTP DDoS attacks while Yadav and Selvakumar [33] noted that HTTP DDoS attacks contain diverse URL and comprise of headers known as user-agent and referrer. The request headers included in HTTP DDoS comprised of an equivalent user-agent including legitimate, incorrect URL and a repeatable request against the equal URL [39].

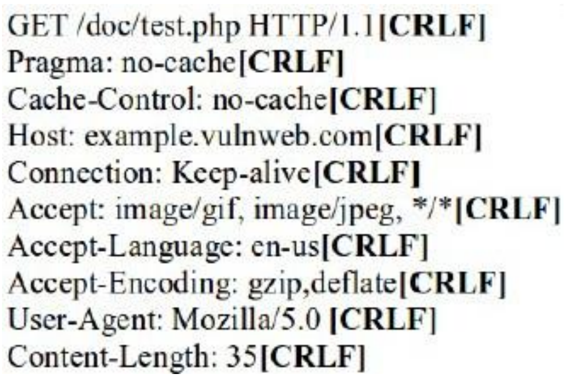
6.2. HTTP DDoS Request Headers (Slow Rate)

The request header patterns for HTTP DDoS in the slow rate category contradicted with request headers employed by session flooding and request flooding attacks as only a single Carriage Return Line Feed (CRLF) exists at the last request headers [16–18]. The genuine request headers are supposed to have two CRLF. The HTTP request transaction contains a request header that will end with a CRLF which refers to the line break of each request header. Yevsieieva and Helalat [17] noted that the appearance of \r\n\r\n shows the headers are complete and indicates the beginning of the body message. The explanation provided in the studies was in line with RFC 2616 guidelines, in which CR refers to \r and LF refers to \n which make the genuine HTTP request to contain \r\n\r\n in the last request headers, indicating the line break of each request headers. Figure 1 illustrates the genuine request headers while Figure 2 indicates fake request headers.



```
GET /doc/test.php HTTP/1.1[CRLF]
Pragma: no-cache[CRLF]
Cache-Control: no-cache[CRLF]
Host: example.vulnweb.com[CRLF]
Connection: Keep-alive[CRLF]
Accept: image/gif, image/jpeg, */*[CRLF]
Accept-Language: en-us[CRLF]
Accept-Encoding: gzip,deflate[CRLF]
User-Agent: Mozilla/5.0 [CRLF]
Content-Length: 35[CRLF][CRLF]
```

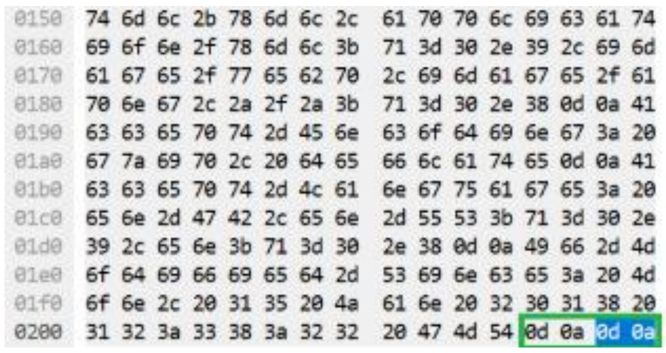
Figure 1. Two CRLF in last request header (Genuine).



```
GET /doc/test.php HTTP/1.1[CRLF]
Pragma: no-cache[CRLF]
Cache-Control: no-cache[CRLF]
Host: example.vulnweb.com[CRLF]
Connection: Keep-alive[CRLF]
Accept: image/gif, image/jpeg, */*[CRLF]
Accept-Language: en-us[CRLF]
Accept-Encoding: gzip,deflate[CRLF]
User-Agent: Mozilla/5.0 [CRLF]
Content-Length: 35[CRLF]
```

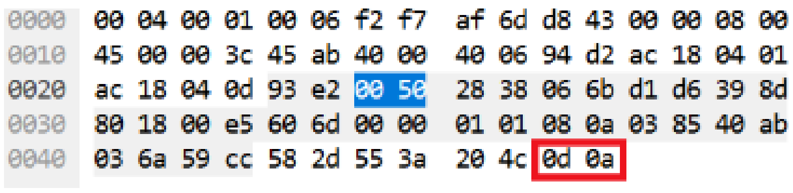
Figure 2. Single CRLF in last request header (Forged).

The authentic CRLF can also be referred as American Standard Code for Information Interchange (ASCII) code that has a value of 0d 0a 0d 0a which represents two CRLF (CRLF, CRLF). However, during the occurrence of HTTP DDoS attack in the slow rate category, only 0d 0a (CRLF) existed [18]. Figure 3 demonstrates the legitimate request headers in ASCII code while Figure 4 indicates the ASCII code for the HTTP DDoS attack.



```
0150  74 6d 6c 2b 78 6d 6c 2c  61 70 70 6c 69 63 61 74
0160  69 6f 6e 2f 78 6d 6c 3b  71 3d 30 2e 39 2c 69 6d
0170  61 67 65 2f 77 65 62 70  2c 69 6d 61 67 65 2f 61
0180  70 6e 67 2c 2a 2f 2a 3b  71 3d 30 2e 38 0d 0a 41
0190  63 63 65 70 74 2d 45 6e  63 6f 64 69 6e 67 3a 20
01a0  67 7a 69 70 2c 20 64 65  66 6c 61 74 65 0d 0a 41
01b0  63 63 65 70 74 2d 4c 61  6e 67 75 61 67 65 3a 20
01c0  65 6e 2d 47 42 2c 65 6e  2d 55 53 3b 71 3d 30 2e
01d0  39 2c 65 6e 3b 71 3d 30  2e 38 0d 0a 49 66 2d 4d
01e0  6f 64 69 66 69 65 64 2d  53 69 6e 63 65 3a 20 4d
01f0  6f 6e 2c 20 31 35 20 4a  61 6e 20 32 30 31 38 20
0200  31 32 3a 33 38 3a 32 32  20 47 4d 54 0d 0a 0d 0a
```

Figure 3. ASCII code for valid HTTP request.



```
0000  00 04 00 01 00 06 f2 f7  af 6d d8 43 00 00 08 00
0010  45 00 00 3c 45 ab 40 00  40 06 94 d2 ac 18 04 01
0020  ac 18 04 0d 93 e2 00 50  28 38 06 6b d1 d6 39 8d
0030  80 18 00 e5 60 6d 00 00  01 01 08 0a 03 85 40 ab
0040  03 6a 59 cc 58 2d 55 3a  20 4c 0d 0a
```

Figure 4. ASCII code for attack HTTP request

7. Analysis of Forged Request Headers

HTTP DDoS comprises many attack strategies as explained in prior studies in Section 4. However, none of the explanations reveal the forged request headers excessively adopted by the attack. HTTP DDoS delivers similar request headers as genuine requests to make the requests look real. Therefore, an extensive analysis is required in order to provide more accurate ingredients when dealing with the attack. The pattern must be understood with the adoption of a current attack script so that the outcome will benefit future studies to understand, propose and enhance the existing HTTP DDoS solution. Section 6.1 disclosed the invalid request headers utilized by HTTP DDoS categories including session flooding and request flooding while Section 6.2 reveals the false request headers utilized by HTTP DDoS categorized as a slow rate. The review of prior work which mentioned about false request headers adopted by HTTP DDoS in Section 6.1 delivers strong indication to mention that the forged request headers produced by HTTP DDoS categories as flooding require extension as past studies provide insufficient information and explanation pertaining to forget request headers adopted by the attack.

8. Materials and Methods

This study adopted a physical hardware when analyzing and disclosing the adoption of request headers in HTTP DDoS. Three attack scripts were executed in the internal network and five attack scripts were executed in an external network to obtain similarities and differences of request headers executed in different network topologies. The attack scripts launched in the public network had capabilities to generate HTTP DDoS traffic through Proxy and Spoof IP address. The attack duration for an internal HTTP DDoS attack was set to 10 min as it is sufficient to acquire the false request headers and investigate the attack pattern due to the high speed of the attack that are capable of generating substantial HTTP requests within a second. Aside from that, for HTTP DDoS through external network, the time period must be lower and was set to 5 min to reduce the impact as the traffic is blocked by ISP and host provider. Behal and Kumar [1] elaborated that evaluation of DDoS in the actual environment reduces the network performance due to plenty of traffic sent by the attack. The required hardware and software in this analysis are shown in Table 1 while Figure 5 illustrates the network architecture.

Table 1. Hardware and software for analysis.

Hardware	Software	Software Function
Web Server Processor Intel(R) Core (TM) i7-6700 CPU @ 3.40GHz 8GB Memory	Python coding editor and Compiler	Edit and execute attack scripts
	Notepad++	View and edit attack scripts
	Proxy Switcher	Identify active public proxies
Client Processor Intel(R) Core (TM) i7-3770 CPU @ 3.40GHz 12GB Memory	Nmap	Identify port usage
	Wireshark	Analyze HTTP request traffics
Firewall	Attack Script	HTTP DDoS Code
	Open Source Operating System: Ubuntu	Provide platform to execute HTTP DDoS attack
	Windows Server 2016 Windows 8 Pro	To host web-based application and initiate HTTP request

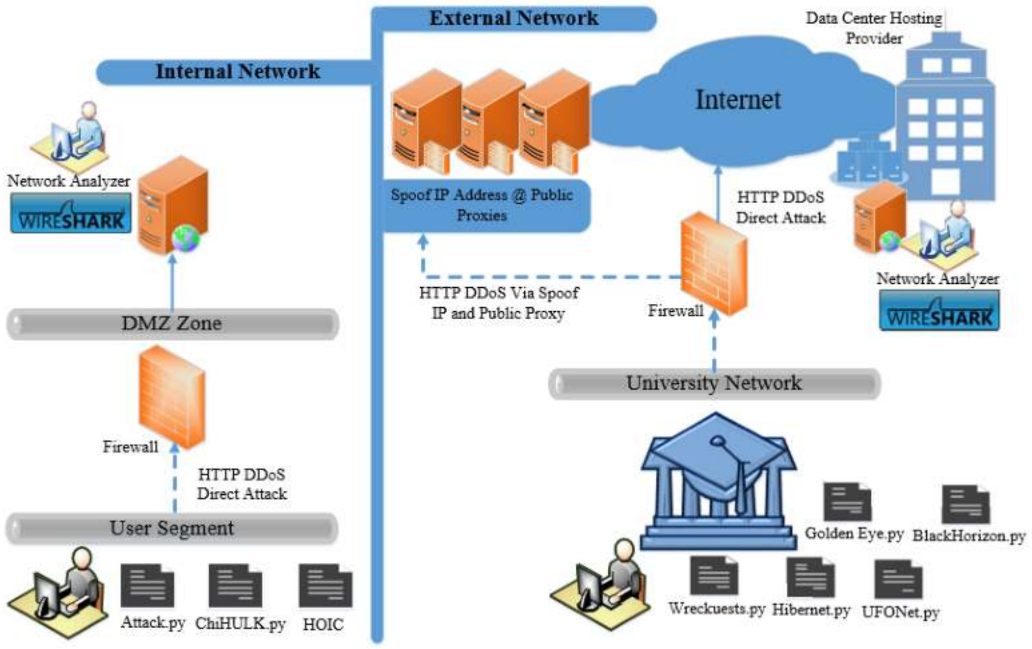


Figure 5. Attack script executed in internal and external networks.

The attack script's execution was conducted separately to ensure the forged request header produced by each attack script can be identified quickly. Execution of the attack in a distributed manner leads to complexity in differentiating malicious and legitimate requests. This approach allows this study to properly elaborate invalid request headers adopted by HTTP DDoS in making the request to appear authentic. Furthermore, HTTP DDoS only requires minimal botnet with the help of an efficient attack script executed in a single machine is sufficient to overloaded a target [21,40].

Ghafar A. Jaafar, et al. [41] explained future studies should utilize a self-generated dataset with the adoption of the attack script which is publicly available. This study utilizes eight attack scripts as indicated in Table 2 to conduct the analysis. Eight attack scripts were deemed as sufficient to conclude forged request headers generated by HTTP DDoS as there are similarities between the scripts in forging the request headers. The critical analysis conducted in Section 8 extensively elaborate similarities and differences between each script. The usage of these attack scripts are publicly available, and have been mentioned and employed broadly by many earlier studies [2,35,36,39,40,42–44]. Although DDoS as services available as noted by Hameed and Ali [45], the usage of attack scripts to be executed in local area network is preferable as DDoS as services require transmission of attack traffic across the network which increases the possibility of the attack traffic to be detected and blocked by many intermediate networks managed by the internet service provider (ISP). The results of the analysis will complement the existing HTTP DDoS request header patterns discovered by prior studies and are thoroughly explained in Sections 6.1 and 6.2

Table 2. Analysis attribute.

No.	Script Name	Attack Scale	Attack Pattern	Attack Duration	Target URL
1.	Attack.py	Internal Attack	High rate direct attack	10 Min	http://lab.com.my
2.	Chihulk.py		High rate direct attack	10 Min Internal	http://lab.com.my
3.	HOIC				
4.	Golden Eye.py	External Attack	High rate direct attack	5 Min	http://42.1.63.189
5.	BlackHorizon.py				
6.	Wreckuests.py	External Attack	High rate through proxy	5 Min	http://42.1.63.189
7.	Hibernet.py				
8.	UFONet.py				

9. Execute Attack Scripts

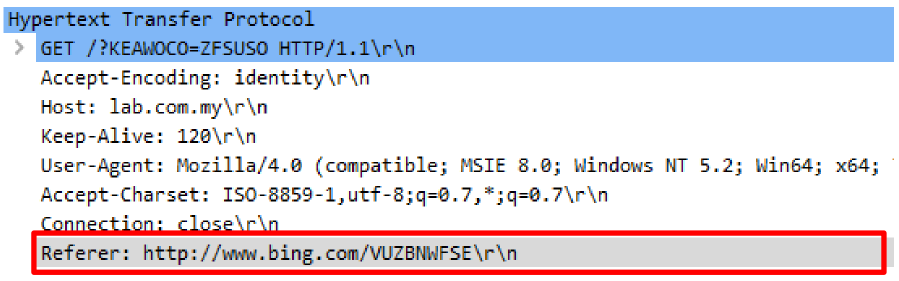
This section highlights the request headers adopted by HTTP DDoS attacks. The request headers that were utilized such as user-agent, referral, query and other request headers that were involved during the genuine request transaction that existed in the HTTP DDoS are disclosed.

9.1. Analysis 1–Attack.py

An attack script known as Attack.py contains 680 false HTTP requests which is nearly similar with genuine request headers. An initial analysis found that the attack scripts adopted random user-agents. The attack script was executed by using the python Attack.py -t 10000 -c 20 http://lab.com.my command. Various user-agents were employed which caused complexity in distinguishing the authenticity of the HTTP request. Further analysis found that the attack randomly constitutes the same request query to a web server. Huge request queries were created with a combination of capital letters to overwhelm a web server. Aside from that, the attack script also comprised of irrelevant HTTP referrals which are not relevant to be the web server's URL source. A valid HTTP referral typically originates from the right sources such as from Google search engine or other relevant sites that are linked to the web server's content. Table 3 illustrates the false user-agent and request query followed by Figures 6 and 7 that indicate the false referrer generated by the attack script.

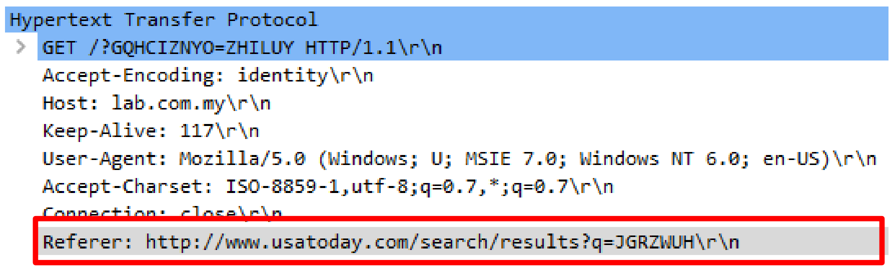
Table 3. User Agent and Request Query.

No.	User Agent String	Request Query
1.	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Win64; x64; Trident/4.0)	KEAWOCO = ZFSUSO
2.	Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)	QJCQABP=MIGMQXRML
3.	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
4.	Mozilla/4.0 (compatible; MSIE 6.1; Windows XP)	DYH=GFOUW
5.	Opera/9.80 (Windows NT 5.2; U; ru) Presto/2.5.22 Version/10.51	GQHCIZNYO=ZHILUY
6.	Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)	DYH=GFOUW



```
Hypertext Transfer Protocol
 > GET /?KEAWOCO=ZFSUSO HTTP/1.1\r\n
   Accept-Encoding: identity\r\n
   Host: lab.com.my\r\n
   Keep-Alive: 120\r\n
   User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Win64; x64;
   Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
   Connection: close\r\n
   Referer: http://www.bing.com/VUZBNWFSE\r\n
```

Figure 6. Forged referral from www.big.com search engine.



```
Hypertext Transfer Protocol
 > GET /?GQHCIZNYO=ZHILUY HTTP/1.1\r\n
   Accept-Encoding: identity\r\n
   Host: lab.com.my\r\n
   Keep-Alive: 117\r\n
   User-Agent: Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)\r\n
   Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
   Connection: close\r\n
   Referer: http://www.usatoday.com/search/results?q=JGRZWUH\r\n
```

Figure 7. Forged referral from www.usatoday.com new web site.

9.2. Analysis 2–Chihulk.py

An attack script known as Chihulk.py was executed by using the Python Chihulk.py http://lab.com.my command. This attack script generated 88769 lines of forged HTTP requests which contain additional user agents to create forged user agents such as mobile devices, web crawler, PlayStation, and open source operating. The existence of user agents known as the web crawlers such as bingbot from MSN and Googlebot from Google indicates that the web server's contents have been indexed by the search engine which will increase the trust that the source of the request originated from a genuine requestor. Further analysis revealed that the attack script generated a request query with a combination of numbers and segregated by a backslash besides being repeatedly sent and continuously generated. Table 4 indicates the forged user agent generated by the attack script and the request query.

Table 4. Fake user agent.

No.	User Agent String	Request Query
1.	Mozilla/5.0 (iPad; U; CPU OS 3_2 like Mac OS X; en-us) AppleWebKit/531.21.10 (KHTML, like Gecko) Version/4.0.4 Mobile/7B334b Safari/531.21.10	\357\277\275\357\277\275{\357\277\275\177 =\357\277\275\357\277\275y
2.	BlackBerry8300/4.2.2 Profile/MIDP-2.0 Configuration/CLDC-1.1 VendorID/107 UP.Link/6.2.3.15.0	\357\277\275\357\277\275\357\277\275 =\357\277\275\357\277\275\357\277\275
3.	BlackBerry9000/5.0.0.93 Profile/MIDP-2.0 Configuration/CLDC-1.1 VendorID/179	y\357\277\275\357\277\275\357\277\275\357\277\275\357\277\275 {\357\277\275=\357\277\275\357\277\275\357\277\275
4.	Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)	~\357\277\275\357\277\275z}\357\277\275 {~=\357\277\275\357\277\275\357\277\275\357\277\275
5	Googlebot/2.1 (http://www.googlebot.com/bot.html)	}\357\277\275\357\277\275=\357\277\275\357\277 \275z\357\277\275\357\277\275~\357\277\275
6.	Mozilla/5.0 (PLAYSTATION 3; 2.00)	z\357\277\275\357\277\275\357\277\275y=\357\277\275

Further explorations successfully reveal that the request query and the HTTP referral were adopted to generate false HTTP requests. The request query appears to be genuine as the accurate format was generated. However, the request query's value is suspicious due to its disconnectedness to the web server's content. Although the value of HTTP referral is the correct URL, it is irrelevant to be accessible from the location. Table 5 shows the HTTP referral attack with the request query while Table 6 presents the comparison between authentic and illegitimate HTTP referrals that were attached with the request query.

Table 5. HTTP referrer with request query.

No	HTTP Referral with Request Query
1.	http://filehippo.com/search?q=%5C221y%5C203%5C231%5C213%7B%5C214%5C217%5C222%5C215%5Cr%5Cn
2.	http://taginfo.openstreetmap.org/search?q=~\177\235z\227\236\r\n
3.	http://www.baoxaydung.com.vn/news/vn/search&q=\225\211\224\235|\240\227\215\216\r\n
4.	https://steamcommunity.com/market/search?q=%5C217x%5C205%5C203%5C226%5C235%7B%5Cr%5Cn
5.	https://www.npmjs.com/search?q=%5C212%5C205%5C207x%7D%5C220%5C232%5C217%5C217%5Cr%5Cn

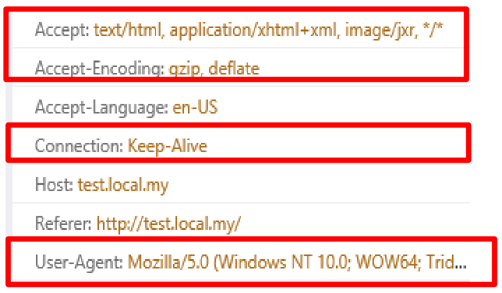
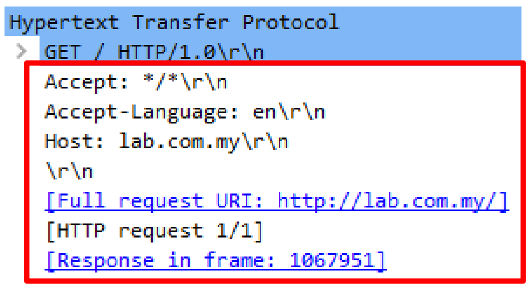
Table 6. Valid and invalid HTTP referrer with request query.

Authentic	Fake
http://ytmnd.com/search?q=catch+that+man\r\n	http://ytmnd.com/search?q=\215\231|\211~\216\230\r\n
http://millercenter.org/search?search=president\r\n	http://millercenter.org/search?q=%5C214%7B%7B%5C205%5C240%5C220%5C211%5Cr%5Cn

9.3. *Analysis 3–High Orbit Ion Cannon (HOIC)*

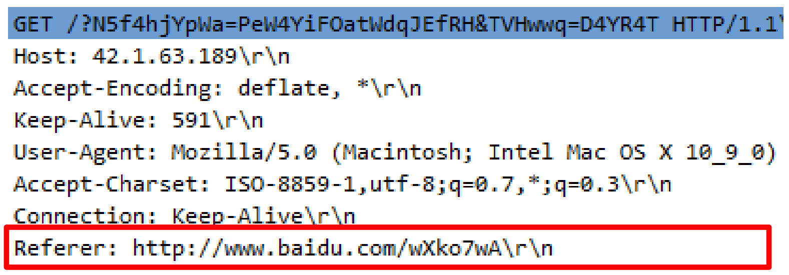
This attack script generated 97,332 false HTTP requests. This attack script focused on generating plenty HTTP requests without completely replicating genuine request headers. These attack scripts contain minimal false request headers such as Accept, Accept-Language and Host which are contradictory to authentic request headers which have more headers. A comparison with legitimate HTTP requests indicates that a complete header was supplied, which contradicts with false request headers generated by the attack script which only had minimal headers. Table 7 indicates the comparison between the complete and incomplete request headers.

Table 7. Complete and incomplete request headers.

Complete GET Header (Genuine)	Incomplete GET Header (HTTP DDoS)
 Accept: text/html, application/xhtml+xml, image/jxr, */* Accept-Encoding: gzip, deflate Accept-Language: en-US Connection: Keep-Alive Host: test.local.my Referer: http://test.local.my/ User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trid...	 Hypertext Transfer Protocol GET / HTTP/1.0\r\n Accept: */*\r\n Accept-Language: en\r\n Host: lab.com.my\r\n \r\n [Full request URI: http://lab.com.my/] [HTTP request 1/1] [Response in frame: 1067951]

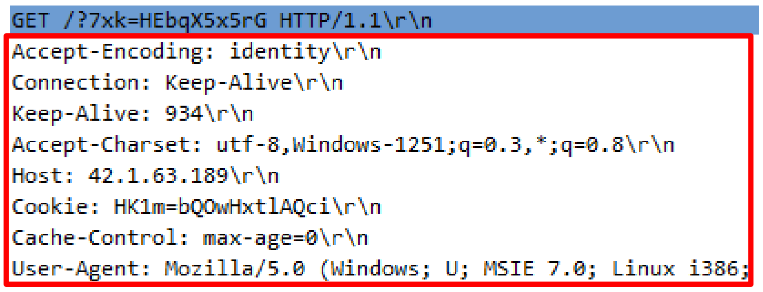
9.4. *Analysis 4–Golden Eye.py*

Another attack script referred as the Golden Eye.py is required to run by using the python GoldenEye.py http://42.1.63.189. The attack scripts generated 21257 forged HTTP requests that were identical to legitimate request headers. The attack scripts constituted almost similar request headers delivered by genuine HTTP request such as user-agent, accept-encoding, connection and referrer. However, the value of HTTP referral cannot be regarded as the URL source as they are unrelated to the web server content and appears to be inconsistent. Figure 8 indicates the irrelevant referral while Figure 9 shows the missing referrer that generated from the same source of HTTP request.



```
GET /?N5f4hjYpWa=PeW4YiFOatWdqJEfRH&TVHwwq=D4YR4T HTTP/1.1
Host: 42.1.63.189\r\n
Accept-Encoding: deflate, *\r\n
Keep-Alive: 591\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_0)
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3\r\n
Connection: Keep-Alive\r\n
Referer: http://www.baidu.com/wXko7wA\r\n
```

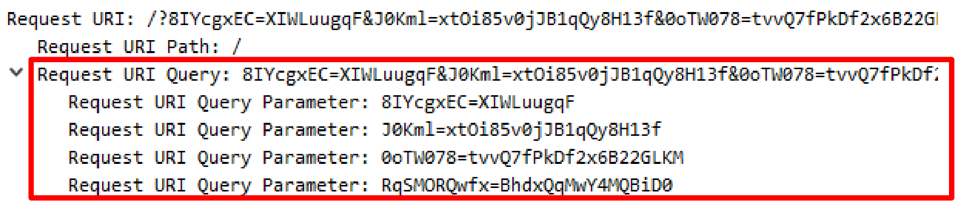
Figure 8. Request headers with referrer.



```
GET /?7xk=HEbqX5x5rG HTTP/1.1\r\n
Accept-Encoding: identity\r\n
Connection: Keep-Alive\r\n
Keep-Alive: 934\r\n
Accept-Charset: utf-8,Windows-1251;q=0.3,*;q=0.8\r\n
Host: 42.1.63.189\r\n
Cookie: HK1m=bQOwHxtlAQci\r\n
Cache-Control: max-age=0\r\n
User-Agent: Mozilla/5.0 (Windows; U; MSIE 7.0; Linux i386;
```

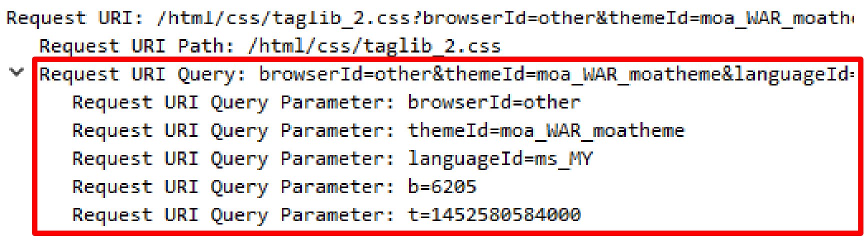
Figure 9. Request headers with missing referrer.

To disclose further how the attack works, an extensive analysis is needed, which successfully reveals that the attack script creates a false longer request query with a combination of small letters, capital letters, symbols and numbers. There are important things that need to be highlighted, for example, the request query constituted by humans began with a question mark (?) and is meaningful compared to the request query generated by the server to fetch items started with the symbol (/). Figure 10 demonstrates the request query generated by the attack script followed by Figure 11 which indicates the request query created by the web server to fetch items.



```
Request URI: /?8IYcgxEC=XIWLuugqF&J0Kml=xtOi85v0jJB1qQy8H13f&0oTW078=tvvQ7fPkDf2x6B22G
  Request URI Path: /
Request URI Query: 8IYcgxEC=XIWLuugqF&J0Kml=xtOi85v0jJB1qQy8H13f&0oTW078=tvvQ7fPkDf
    Request URI Query Parameter: 8IYcgxEC=XIWLuugqF
    Request URI Query Parameter: J0Kml=xtOi85v0jJB1qQy8H13f
    Request URI Query Parameter: 0oTW078=tvvQ7fPkDf2x6B22GLKM
    Request URI Query Parameter: RqSMORQwfx=BhdxQqMwY4MQBiD0
```

Figure 10. False request query in DTS3.

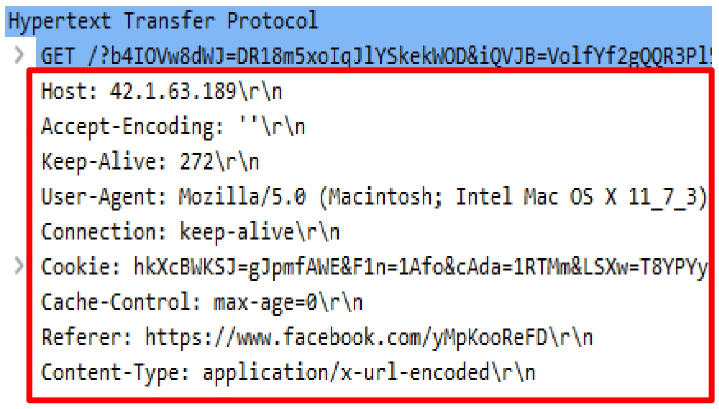


```
Request URI: /html/css/taglib_2.css?browserId=other&themeId=moa_WAR_moath
  Request URI Path: /html/css/taglib_2.css
Request URI Query: browserId=other&themeId=moa_WAR_moatheme&languageId
    Request URI Query Parameter: browserId=other
    Request URI Query Parameter: themeId=moa_WAR_moatheme
    Request URI Query Parameter: languageId=ms_MY
    Request URI Query Parameter: b=6205
    Request URI Query Parameter: t=1452580584000
```

Figure 11. Genuine request query generates by server.

9.5. *Analysis 5–BlackHorizon.py*

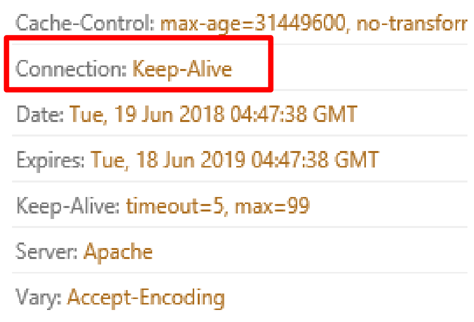
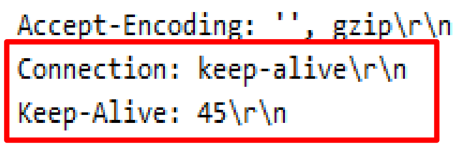
The BlackHorizon.py requires the python BlackHorizon.py http://42.1.63.189 command to execute; and it generated 23,974 invalid HTTP requests which are likely to appear as legitimate request. Initial inquiries indicate that all the required request headers were supplied. However, a detailed analysis shows that the attack script comprises afalse request query which contains capital letters, small letters, numbers, and symbols. Further analysis revealed that the existence of an incorrect header in HTTP requests such as Keep-Alive has been found. The Keep-Alive is the value of the header in HTTP responses. However, it has been missused by attackers to be attached to the header. Figure 12 illustrates the request header supplied by the attack script while Table 8 indicates a comparison of valid and invalid KeepAlive values which involve in HTTP request.



```
Hypertext Transfer Protocol
> GET /?b4IOVw8dWJ=DR18m5xoIqJlYSkekWOD&iQVJB=VolfYf2gQQR3Pl
  Host: 42.1.63.189\r\n
  Accept-Encoding: ''\r\n
  Keep-Alive: 272\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 11_7_3)
  Connection: keep-alive\r\n
> Cookie: hkXcBWKSJ=gJpmfAWE&F1n=1Afo&cAda=1RTMm&LSXw=T8YPYy
  Cache-Control: max-age=0\r\n
  Referer: https://www.facebook.com/yMpKooReFD\r\n
  Content-Type: application/x-url-encoded\r\n
```

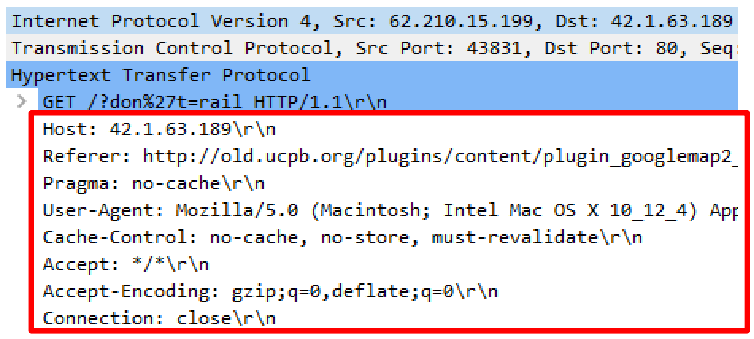
Figure 12. DTS5 request headers.

Table 8. Valid and invalid Keep-Alive values.

Valid Keep Alive from HTTP Response	Invalid Keep-Alive from GET Header
 Cache-Control: max-age=31449600, no-transforr Connection: Keep-Alive Date: Tue, 19 Jun 2018 04:47:38 GMT Expires: Tue, 18 Jun 2019 04:47:38 GMT Keep-Alive: timeout=5, max=99 Server: Apache Vary: Accept-Encoding	 Accept-Encoding: '', gzip\r\n Connection: keep-alive\r\n Keep-Alive: 45\r\n

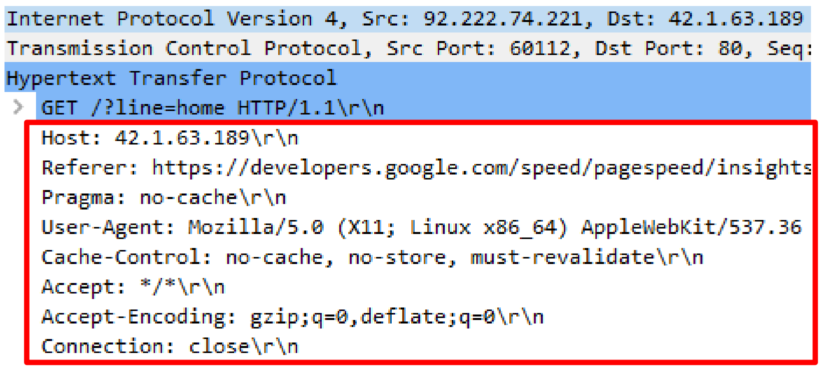
9.6. Analysis 6–Wreckuests.py

The Wreckuests.py attack script generated the HTTP DDoS traffic through spoof IP address and required the python3 Wreckuests.py -v http://42.1.63.189189 command to execute. The attack script generated 9397 false HTTP requests that were sent through spoof IP address. Initial analysis disclosed that the attack script generated meaningful request query which can be understood by humans even when the query was unrelated to the web server's content. Further analysis revealed that the source of the spoof IP address and software known as ZenMap were utilized. Three IP addresses were opted randomly, and the outcomes indicate that IP address 62.210.15.199 belongs to a web server and utilized port 22, 80, 443 and 3306 to operate. Next, IP address 92.222.74.221 was selected and the result showed that the IP address was assigned to a web server that utilized port 22 and 80 to operate. Finally, IP address 103.234.254.10 was analyzed and the outcome indicated that the IP address is owned by a router and adopted port 53 and 1723 to operate. The usage of this attack script indicates that the source of HTTP DDoS attack can also originate from IP spoofing such as the server and router. Figures 13–16 indicate the request headers that were made by the three IP addresses discussed earlier while Figures 17 and 18 illustrate the IP address verification.



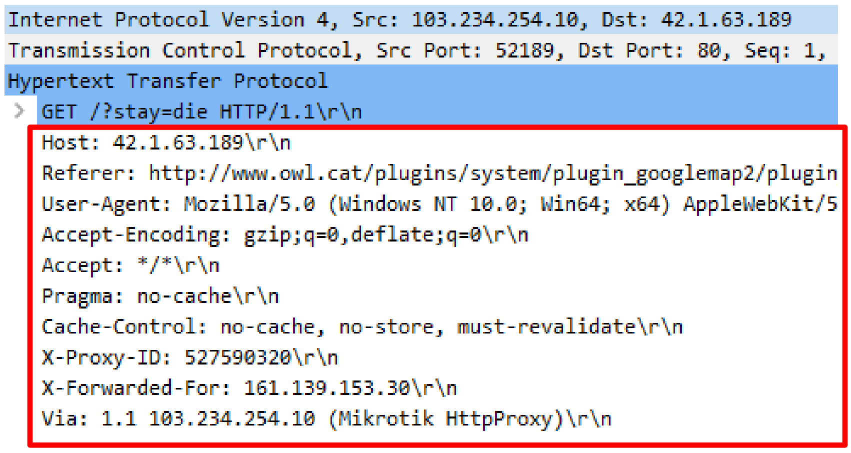
```
Internet Protocol Version 4, Src: 62.210.15.199, Dst: 42.1.63.189
Transmission Control Protocol, Src Port: 43831, Dst Port: 80, Seq
Hypertext Transfer Protocol
 > GET /?don%27t=rail HTTP/1.1\r\n
   Host: 42.1.63.189\r\n
   Referer: http://old.ucpb.org/plugins/content/plugin_googlemap2_
   Pragma: no-cache\r\n
   User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4) App
   Cache-Control: no-cache, no-store, must-revalidate\r\n
   Accept: */*\r\n
   Accept-Encoding: gzip;q=0,deflate;q=0\r\n
   Connection: close\r\n
```

Figure 13. Request headers generate by IP address 62.210.15.199.



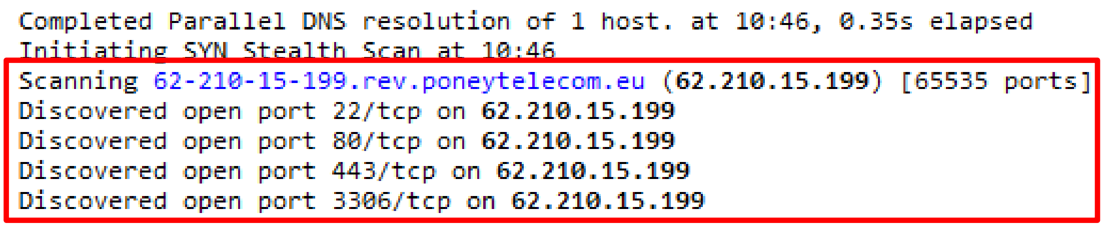
```
Internet Protocol Version 4, Src: 92.222.74.221, Dst: 42.1.63.189
Transmission Control Protocol, Src Port: 60112, Dst Port: 80, Seq:
Hypertext Transfer Protocol
 > GET /?line=home HTTP/1.1\r\n
   Host: 42.1.63.189\r\n
   Referer: https://developers.google.com/speed/pagespeed/insights
   Pragma: no-cache\r\n
   User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
   Cache-Control: no-cache, no-store, must-revalidate\r\n
   Accept: */*\r\n
   Accept-Encoding: gzip;q=0,deflate;q=0\r\n
   Connection: close\r\n
```

Figure 14. Request headers generate by IP address 92.222.74.221.



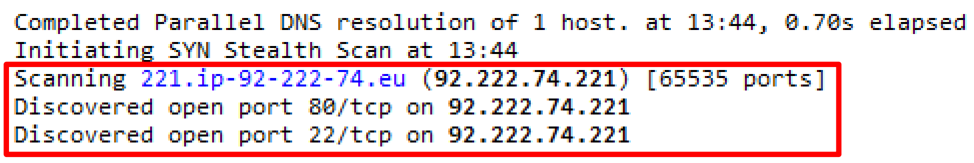
```
Internet Protocol Version 4, Src: 103.234.254.10, Dst: 42.1.63.189
Transmission Control Protocol, Src Port: 52189, Dst Port: 80, Seq: 1,
Hypertext Transfer Protocol
 > GET /?stay=die HTTP/1.1\r\n
   Host: 42.1.63.189\r\n
   Referer: http://www.owl.cat/plugins/system/plugin_googlemap2/plugin
   User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/5
   Accept-Encoding: gzip;q=0,deflate;q=0\r\n
   Accept: */*\r\n
   Pragma: no-cache\r\n
   Cache-Control: no-cache, no-store, must-revalidate\r\n
   X-Proxy-ID: 527590320\r\n
   X-Forwarded-For: 161.139.153.30\r\n
   Via: 1.1 103.234.254.10 (Mikrotik HttpProxy)\r\n
```

Figure 15. Request headers generate by IP address 103.234.254.10.



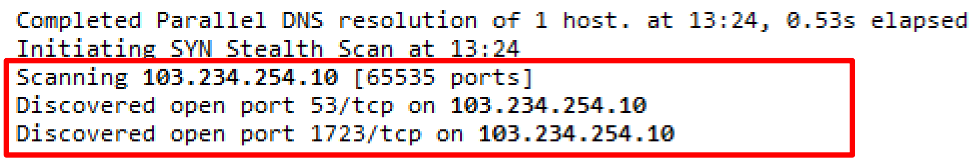
```
Completed Parallel DNS resolution of 1 host. at 10:46, 0.35s elapsed
Initiating SYN Stealth Scan at 10:46
Scanning 62-210-15-199.rev.poneytelecom.eu (62.210.15.199) [65535 ports]
Discovered open port 22/tcp on 62.210.15.199
Discovered open port 80/tcp on 62.210.15.199
Discovered open port 443/tcp on 62.210.15.199
Discovered open port 3306/tcp on 62.210.15.199
```

Figure 16. IP address source verification for 62.210.15.199.



```
Completed Parallel DNS resolution of 1 host. at 13:44, 0.70s elapsed
Initiating SYN Stealth Scan at 13:44
Scanning 221.ip-92-222-74.eu (92.222.74.221) [65535 ports]
Discovered open port 80/tcp on 92.222.74.221
Discovered open port 22/tcp on 92.222.74.221
```

Figure 17. IP address source verification for 92.222.74.221.

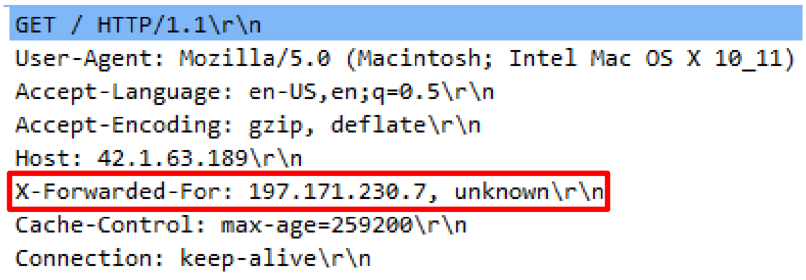


```
Completed Parallel DNS resolution of 1 host. at 13:24, 0.53s elapsed
Initiating SYN Stealth Scan at 13:24
Scanning 103.234.254.10 [65535 ports]
Discovered open port 53/tcp on 103.234.254.10
Discovered open port 1723/tcp on 103.234.254.10
```

Figure 18. IP address source verification for 103.234.254.10.

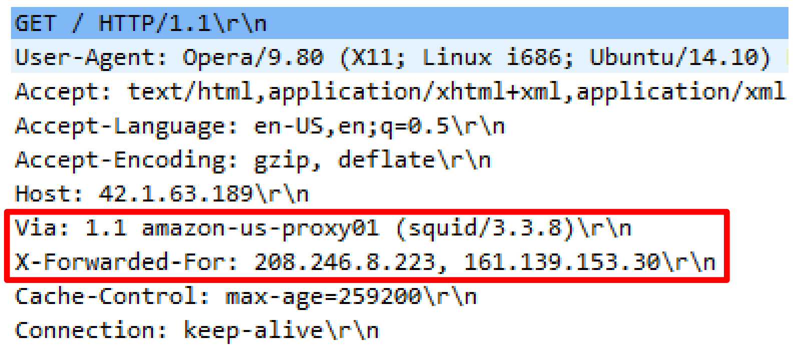
9.7. Analysis 7–Hibernet.py

Attack script recognized as Hibernet.py employed the python3 Hibernet.py http://42.1.63.189 command to operate. The attack script contained 70792 invalid HTTP requests generated through various public proxies to launch an attack over a web server. To successfully execute the attack script, a list of public proxy IP addresses are required and must be located within the proxy file. Initial inquiry indicated that necessary request headers were supplied; however, the existence of proxy headers was inconsistent. Proxy headers attached in the HTTP requests also vary. Firstly, only single proxy header existed such as X-Forwarded-For in one HTTP request. Secondly, double proxy headers known like X-Forwarded-For and VIA, thirdly triple proxy headers such as X-Proxy-ID with X-Forwarded-For and VIA. These patterns show that each proxy provider has different approaches to introduce the source connection from which the proxy originates from when establishing a web server. The various proxy headers utilized by the attack scripts are illustrated in Figures 19–21.



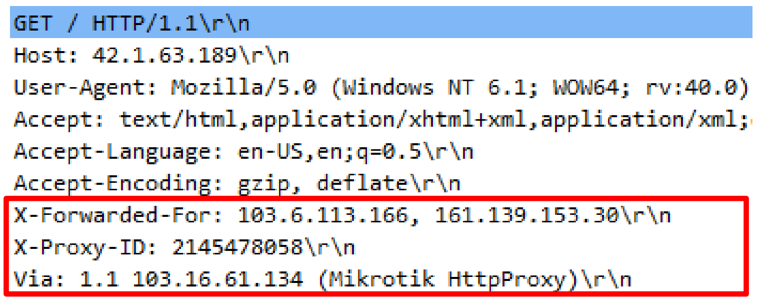
```
GET / HTTP/1.1\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11)
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Host: 42.1.63.189\r\n
X-Forwarded-For: 197.171.230.7, unknown\r\n
Cache-Control: max-age=259200\r\n
Connection: keep-alive\r\n
```

Figure 19. X-Forward-For in request header.



```
GET / HTTP/1.1\r\n
User-Agent: Opera/9.80 (X11; Linux i686; Ubuntu/14.10)
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Host: 42.1.63.189\r\n
Via: 1.1 amazon-us-proxy01 (squid/3.3.8)\r\n
X-Forwarded-For: 208.246.8.223, 161.139.153.30\r\n
Cache-Control: max-age=259200\r\n
Connection: keep-alive\r\n
```

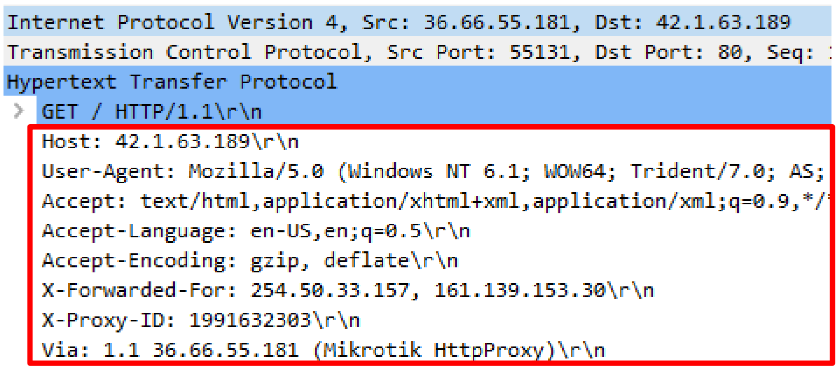
Figure 20. X-Forward-For and Via in request header.



```
GET / HTTP/1.1\r\n
Host: 42.1.63.189\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0)
Accept: text/html,application/xhtml+xml,application/xml;
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
X-Forwarded-For: 103.6.113.166, 161.139.153.30\r\n
X-Proxy-ID: 2145478058\r\n
Via: 1.1 103.16.61.134 (Mikrotik HttpProxy)\r\n
```

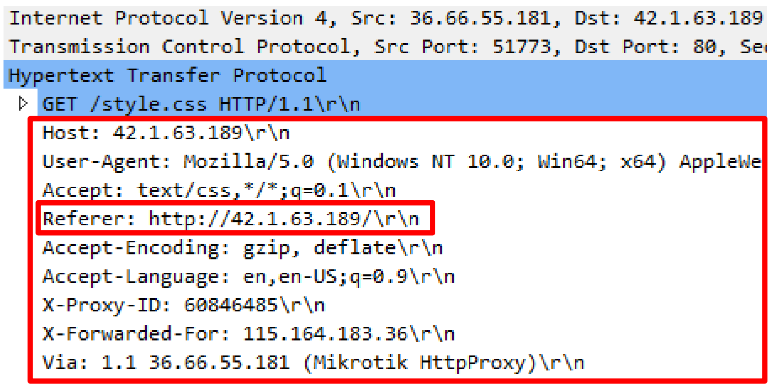
Figure 21. X-Forward-For, Via and X-Proxy-ID in request header.

Results obtained in this study also reveal that some of the HTTP requests were missing. The missing header is known as a referrer which commonly exists in an HTTP request. To ascertain that the headers were missing due to fake HTTP requests, the proxy IP address utilized in the attack script was used to launch a genuine HTTP request. This is to ensure that the missing header was not due to the proxy intentionally hidden the header. The proxy IP address was 36.66.55.181 and utilized port 8080 for clients to connect to the proxy. As a result, a genuine request against a web server using the proxy supplied all the common headers. Figure 22 indicates the header provided by the attack script while Figure 23 illustrates the header that adopted the proxy.



```
Internet Protocol Version 4, Src: 36.66.55.181, Dst: 42.1.63.189
Transmission Control Protocol, Src Port: 55131, Dst Port: 80, Seq:
Hypertext Transfer Protocol
 > GET / HTTP/1.1\r\n
   Host: 42.1.63.189\r\n
   User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; AS;
   Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/
   Accept-Language: en-US,en;q=0.5\r\n
   Accept-Encoding: gzip, deflate\r\n
   X-Forwarded-For: 254.50.33.157, 161.139.153.30\r\n
   X-Proxy-ID: 1991632303\r\n
   Via: 1.1 36.66.55.181 (Mikrotik HttpProxy)\r\n
```

Figure 22. Missing referrer in HTTP request.



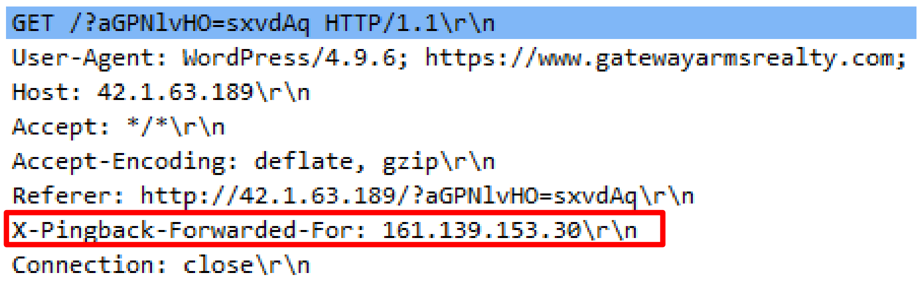
```
Internet Protocol Version 4, Src: 36.66.55.181, Dst: 42.1.63.189
Transmission Control Protocol, Src Port: 51773, Dst Port: 80, Se
Hypertext Transfer Protocol
 ▷ GET /style.css HTTP/1.1\r\n
   Host: 42.1.63.189\r\n
   User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
   Accept: text/css,*/*;q=0.1\r\n
   Referer: http://42.1.63.189/\r\n
   Accept-Encoding: gzip, deflate\r\n
   Accept-Language: en,en-US;q=0.9\r\n
   X-Proxy-ID: 60846485\r\n
   X-Forwarded-For: 115.164.183.36\r\n
   Via: 1.1 36.66.55.181 (Mikrotik HttpProxy)\r\n
```

Figure 23. Complete HTTP request with referrer.

9.8. Analysis 8–UFONet.py

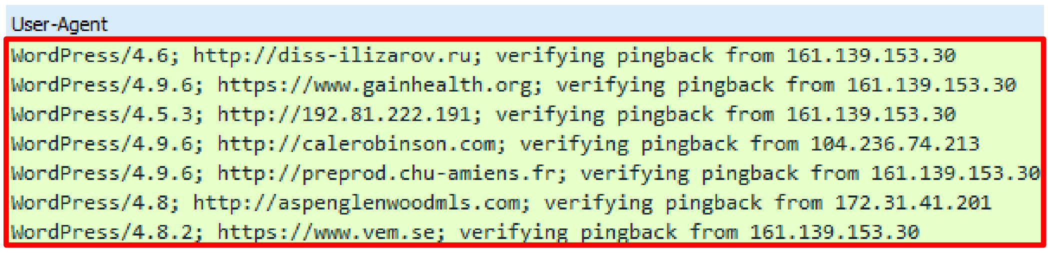
The final attack script executed in this study is the UFONet.py. The attack script generated minimal fake GET requests compared to the previous attack scripts that were executed. The attack script generates 51 HTTP requests which were launched through a proxy to attack a web server. The attack script was operated using the /ufonet -a http://42.1.63.189 -r 10 -threads 500 command.

The preliminary analysis revealed that the attack script adopted a different name, signifying that an HTTP request came from a proxy by using the X-Pingback-Forwarded-For header. Aside from that, the command proxy publicly known as VIA was not present. Detailed analysis disclosed that the attack script generated an odd value for a user agent. User agent conveys client information including the browser name, operating system version, etc. Apart from that, an inconsistent referral header was detected and appeared in each HTTP request. The analysis also found that the attack script generated request queries that were attached together with the referral header along with the URL, thus indicating that the query was generated from the previous page and was forwarded to next page. Figure 24 demonstrates the GET header and the HTTP referral followed by Figure 25 which indicates the list of the user-agent.



```
GET /?aGPNlvHO=sxvdAq HTTP/1.1\r\n
User-Agent: WordPress/4.9.6; https://www.gatewayarmsrealty.com;
Host: 42.1.63.189\r\n
Accept: */*\r\n
Accept-Encoding: deflate, gzip\r\n
Referer: http://42.1.63.189/?aGPNlvHO=sxvdAq\r\n
X-Pingback-Forwarded-For: 161.139.153.30\r\n
Connection: close\r\n
```

Figure 24. Proxy header in DTS8.



```
User-Agent
WordPress/4.6; http://diss-ilizarov.ru; verifying pingback from 161.139.153.30
WordPress/4.9.6; https://www.gainhealth.org; verifying pingback from 161.139.153.30
WordPress/4.5.3; http://192.81.222.191; verifying pingback from 161.139.153.30
WordPress/4.9.6; http://calerobinson.com; verifying pingback from 104.236.74.213
WordPress/4.9.6; http://preprod.chu-amiens.fr; verifying pingback from 161.139.153.30
WordPress/4.8; http://aspenglenwoodmls.com; verifying pingback from 172.31.41.201
WordPress/4.8.2; https://www.vem.se; verifying pingback from 161.139.153.30
```

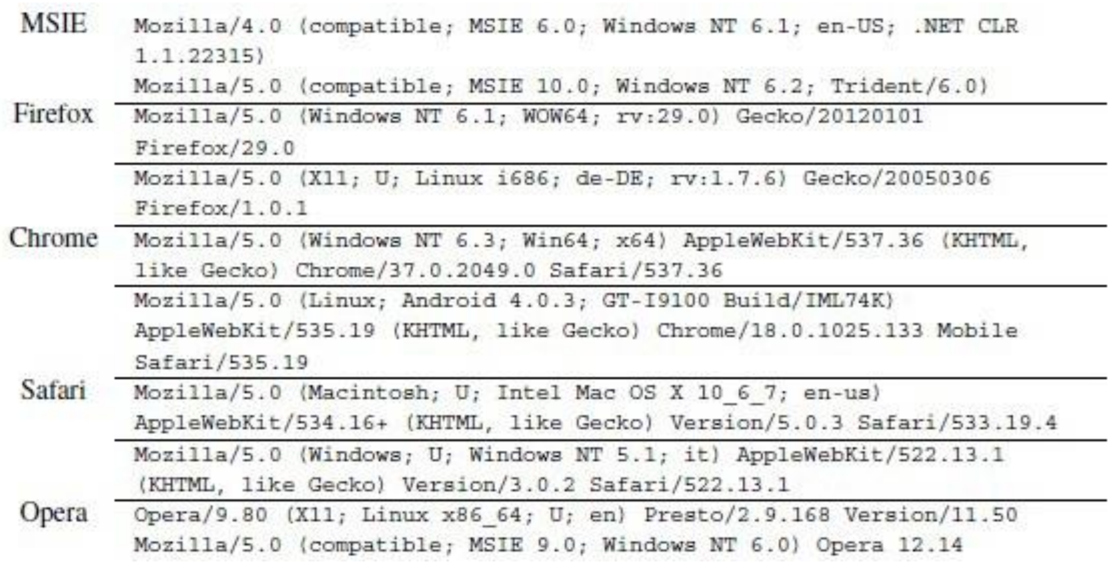
Figure 25. User agent in DTS8.

10. Overview of Critical Analysis

This section elaborates the similarities and differences of the forged request headers constituted by the attack scripts executed in the previous section (Section 9.1 until Section 9.8). The most manipulated request headers will be analyzed critically including the user-agent, query string, referral, connection, accept-language and proxy headers. Although there are numerous request headers available that are involved during a genuine HTTP request transaction, this study advocates that the selection of the header as noted above were suffient to reveal the use of forged request headers when executing HTTP DDoS attacks.

10.1. GET Header-User-Agent

All attack scripts that have been executed contain user-agents. The user-agents delivered in all attack scripts are most likely to be equal as those found in a genuine GET header. However, there are several attack scripts that indicate a different appearance such as the HOIC attack script that supplied a minimal header with the absence of a user-agent. Apart from that, the Chihulk.py attack script adopted a user-agent that comes from a variety of platforms as it contains mobile device name, web crawler from Googlebot and video game devices such as Play Station. In contrast, The UFONet.py attack script delivers an odd user-agent as it comprises the URL and IP address which do not provide information about the requestor. The user agent is expected to supply information about the client including the web browser version, operating system, etc. Zhang, et al. [46] note that information contained in a common web browser. However, this study gathers contrasting findings in which none of the web browsers contain such information as explicated by the results derived from the UFONet.py attack script. Apart from that, Gou, et al. [47] noted that a genuine HTTP request contains headers such as “host”, “connection”, “accept”, “user-agent” or any headers that relate to the request. The user-agent is utilized by the web user’s browser by indicating the information of a request [29]. Figure 26 illustrates the user-agent that is included in the web browser.



MSIE	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 6.1; en-US; .NET CLR 1.1.22315)
	Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Trident/6.0)
Firefox	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20120101 Firefox/29.0
	Mozilla/5.0 (X11; U; Linux i686; de-DE; rv:1.7.6) Gecko/20050306 Firefox/1.0.1
Chrome	Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2049.0 Safari/537.36
	Mozilla/5.0 (Linux; Android 4.0.3; GT-I9100 Build/IML74K) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.133 Mobile Safari/535.19
Safari	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_7; en-us) AppleWebKit/534.16+ (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4
	Mozilla/5.0 (Windows; U; Windows NT 5.1; it) AppleWebKit/522.13.1 (KHTML, like Gecko) Version/3.0.2 Safari/522.13.1
Opera	Opera/9.80 (X11; Linux x86_64; U; en) Presto/2.9.168 Version/11.50
	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.0) Opera 12.14

Figure 26. User-Agent in Web Browser.

10.2. GET Header–Request Query

The request query supplied by each attack script was contradictory to a genuine request as the attacker tried to mimic the request made by humans. The request query included in the Attack.py attack script was created using capital letters while the Chihulk.py attack script comprised of a number and a combination of few symbols. Emulating a legitimate request does not necessarily supply a request query as proven in the HOIC and Hibetnet.py attack scripts. The Golden Eye.py and BlackHorizon.py attack scripts adopted longer request queries constituting a combination of small letters, capital letters, numbers and symbols. Besides that, there is no predefined limit for HTTP request to process request headers and its values. Furthermore, an appropriate length for request header is difficult to be defined [48]. Request queries produced by the Wreckuests.py attack script delivered a brilliant strategy in mimicking a request query as it contains a query that can be understood by humans although it is irrelevant for a web server to process. Besides that, queries found in other attack scripts are not readable. Request query generated by the UFONet.py attack script contains query that is identical to Golden Eye.py and BlackHorizon.py attack scripts. However, the query is found to be shorter. It should be noted that the web server adopted in this analysis was designed with a static HTML page and was not designed to process a query. Hence, the continuous request query received has proven that the GET request is malicious.

10.3. GET Header-Referral

Almost all attack scripts supplied a referral header to show that the current web pages were accessible from the previous web page. According to Reid [49], a header referrer refers to the earlier address for web page browsed by users. In this context, any web addresses that belong to any website are permitted to be a referral as the existence of the referral address in terms of its relevance was not examined. This analysis has been successful at disclosing that all attack scripts have attached a referral with a valid URL address with the exception of the HOIC attack script. However, the source of the address is irrelevant to be a value for referral header for the current page that is being accessed.

This pattern shows that the HTTP referral has been manipulated. Fielding, et al. [50] suggest that for HTTP version 1.1, a user has an option to select whether the referral header is sent as the referral which may contain a private link and the presence of a referral in an HTTP request may reveal confidential information. In terms of protocol showed in the referral header, all attack scripts utilize an HTTP protocol except for Blackhorizon.py attack script as it adopted a referral from an HTTPS protocol that contained Facebook address, which provides an understanding that a web page is accessible from the previous web page which is from Facebook.

Most of the attack scripts generated referrals that come from a search engine to show that the current web page was previously searched and is accessible from a search engine which are commonly performed by genuine users. On the other hand, the referral is supposed to be encrypted as it has the ability to reveal the web browsing history and information that belonged to the user that may exist in these headers [51]. Dolnak [52] addresses the problem highlighted by Fielding and Reschke [51] where they suggested a referrer header policy that can control information indicated by the referrer to reduce the risk of information leakage. However, the proposed solution does not curb HTTP referrer manipulation created by HTTP DDoS attacks.

10.4. GET Header-Connection and Accept Language

HTTP requests contain standard value and header which have been defined by the International body. However, the existence of incorrect headers and value signal that the GET header has been manipulated. During the execution of Attack.py and Chihulk.py attack scripts, the connection header marks a close which entails that a web server has stopped receiving GET requests. However, in the case on these attack scripts, the header marks a close with the web server receiving repetitive HTTP requests. Besides, the Keep-Alive is supposed to be the header for an HTTP response and the value for a connection header [53]. The appearance of Keep-Alive in the request header produced by *Golden* Eye.py and BlackHorizon.py attack scripts indicates that attackers accidentally use the header to emulate a user's request when executing HTTP DDoS attacks. In addition, the Wreckuests.py, Hibernet.py and UFONet.py attack scripts do not attach Keep-Alive as the header while the HOIC attack script contains a minimal GET header with the absence of a connection header.

The Accept-Language in a request header indicates the web browser language used by users and the existence of this header in an HTTP request can be used to identify the geolocation of users [49]. This study found that the Accept-Language header was not generated by five attack scripts (Attack.py, Chihulk.py, GoldenEye.py, BlackHorizon.py and Wreckuests.py). Three attack scripts (GoldenEye.py, Hibernet.py and UFONet.py) indicate an inconsistent appearance as the header was missing for certain HTTP requests. The Accept-Language shows the language utilized by users in web browser and the missing of this header indicate that the source initiators of an HTTP request was generated from automated tools. Clients adopt web browsers to browse through the web server content, which will create an HTTP request. The Connection and Accept-Language headers are the general headers that exist in an HTTP request [47].

10.5. Proxy Headers and IP Spoofing

The HTTP request will provide proxy headers if the source requestor adopted a proxy. The Hibernet.py and UFONet.py attack scripts comprise of many proxies, indicating that proxy headers were not displayed by certain proxy providers. Another method utilized by the attacker was IP spoofing to execute HTTP DDoS as adopted by the Wreckuests.py attack script. The usage of IP address spoofing to launch the attack poses difficulty as the attacker is capable of attaching the proxy header thereby making the request to appear as if it comes from proxy.

The Hibernet.py attack script generated two IP addresses in X-Forwarded-For headers in which the second IP address belongs to this study's equipment and the first IP is believed to originate from another proxy. Petersson and Nilsson [54] noted that a proxy require to establish a connection to another proxy when it is necessary to acquire the web server content and utilize headers known as X-Forwarded-For.

The UFONet.py attack script delivers a proxy header named as X-Pingback-Forwarded-For which does not equal to the standard proxy header known as X-Forwarded-For. The findings support the explanation made by Petersson and Nilsson [54]. In this study, the existence of distinct proxy headers and the inconsistent appearances due to proxy header are not updated by proxy provider and that the headers are optional to be displayed in the GET request header.

11. Results

Based on the execution of eight attack scripts, this analysis has successfully discloses seven (7) forged request headers patterns produced by HTTP DDoS. The request header pattern for each attack script executed in this study has been thoroughly discussed in Section 9.1 to Section 9.8. Results of the study are further supported by a critical analysis reported in the previous section (Section 10.1 to Section 10.5) which provide a strong justification to assert that HTTP DDoS has a specific pattern to generate forged HTTP requests. This study's findings corroborated with the findings obtained in prior studies with regard to the request header patterns, which pave the way for future studies to apply appropriate approach when dealing HTTP DDoS attack. HTTP DDoS attacks are capable of generating a substantial request in a minimal duration as the source of the request originates from various platforms. The use of attack scripts in this study comprises HTTP DDoS from a direct attack, both through proxy and spoof IP address. Hence, based on the attack scripts utilized in this study, the architecture of HTTP DDoS has been successfully designed. Additionally, the taxonomy of the forged request headers has also been designed including false request headers mentioned in prior studies described in Section 5. The architecture of HTTP DDoS attack is illustrated in Figure 27 followed by descriptions regarding forged GET headers produced by HTTP DDoS attacks. The taxonomy of the forged request headers is depicted in Figure 28.

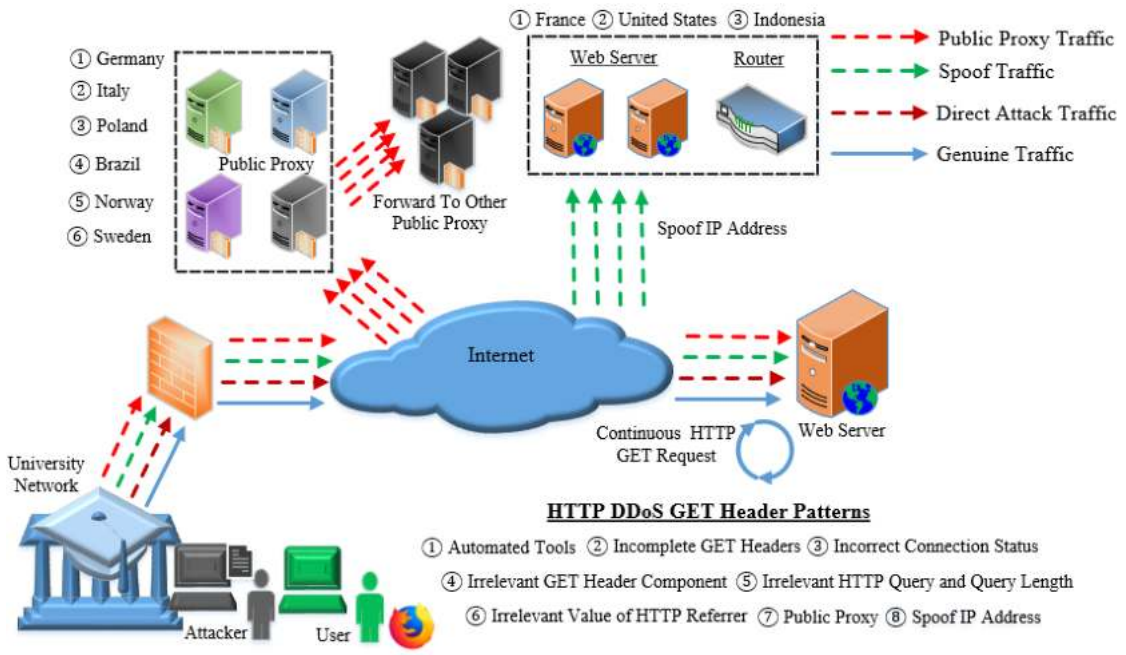

① France ② United States ③ Indonesia
Web Server
Router
Public Proxy Traffic
Spoof Traffic
Direct Attack Traffic
Genuine Traffic
① Germany
② Italy
③ Poland
④ Brazil
⑤ Norway
⑥ Sweden
Public Proxy
Forward To Other Public Proxy
Spoof IP Address
Internet
Continuous HTTP GET Request
Web Server
University Network
Attacker
User
HTTP DDoS GET Header Patterns
① Automated Tools ② Incomplete GET Headers ③ Incorrect Connection Status
④ Irrelevant GET Header Component ⑤ Irrelevant HTTP Query and Query Length
⑥ Irrelevant Value of HTTP Referrer ⑦ Public Proxy ⑧ Spoof IP Address

Figure 27. Architecture of HTTP DDoS attack.

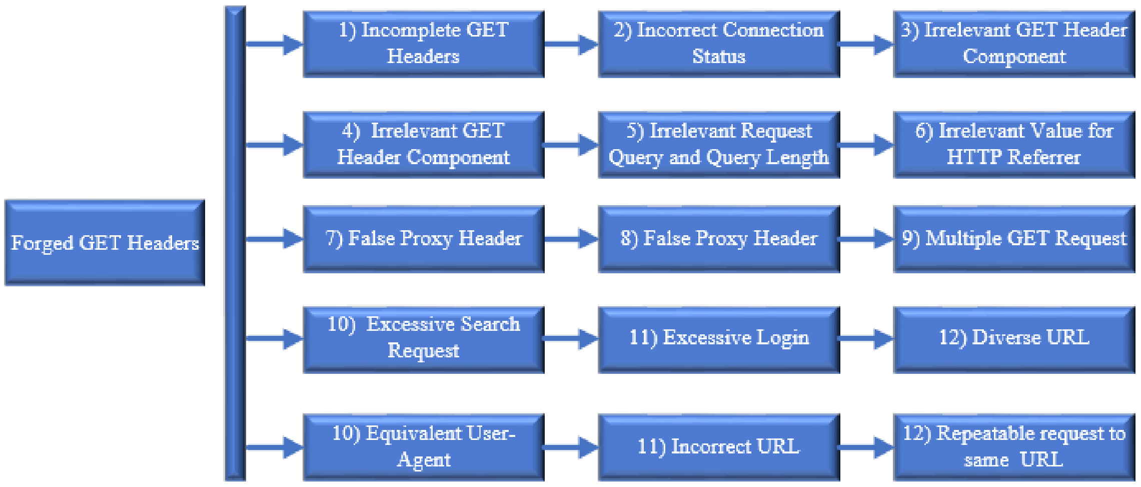

Forged GET Headers
1) Incomplete GET Headers
2) Incorrect Connection Status
3) Irrelevant GET Header Component
4) Irrelevant GET Header Component
5) Irrelevant Request Query and Query Length
6) Irrelevant Value for HTTP Referrer
7) False Proxy Header
8) False Proxy Header
9) Multiple GET Request
10) Excessive Search Request
11) Excessive Login
12) Diverse URL
10) Equivalent User-Agent
11) Incorrect URL
12) Repeatable request to same URL

Figure 28. Taxonomy of Forged GET headers by HTTP DDoS attack.

- **Automated Tools:** HTTP DDoS attack requires enormous HTTP requests to overwhelm web servers. Hence, attackers are required to adopt automated tools to generate massive requests which only require a minimal duration to create forged request traffics. A massive GET requests received results in the web server to become unresponsive due to overloading as it reaches beyond its capacity to process HTTP request.
- **Incomplete Request Headers:** HTTP DDoS attacks generate an incomplete request header as it is unnecessary to supply complete headers as the purpose of the attack is to gain a server's resources to cause the web server to be unresponsive. Attackers are able to generate substantial HTTP requests without a request header. However, to create complexities in detecting the attack, attackers must supply the request headers to emulate user's access pattern to avoid from being detected.
- **Incorrect Connection Status:** The connection status header in an HTTP request presents the status of the client which has an active connection with a web server. However, during an occurrence of HTTP DDoS attack, the status is marked as close with continuous HTTP requests received. Apart from that, the existence of Keep-Alive in the request header is a sign of HTTP DDoS as it is not the header for an HTTP request.
- **Irrelevant Request Header Component:** The existence of request headers in an HTTP request has been explained in the request for comments (RFC) which is hosted by the Internet Engineering Task Force (IETF). Hence, the appearance of non-standard request headers in a repeated HTTP request provides a strong indicator that the source of the request originates from an attacker.
- **Irrelevant Request Query and Query Length:** A client will search for information that relates to the web server's content and adopts a human language. This search will then be translated into a request query to forward the search to the web server to be processed. An HTTP DDoS attack emulates this process by creating a request query which contains either special characters such as '!', '@', '#', '$', and '^' or a longer query.
- **Irrelevant Value for HTTP Referrer:** The value for HTTP referrer is the previous URL accessed by a user. An HTTP DDoS attack is capable of mimicking this operation by attaching a valid URL. Although the attack is able to supply the valid URL, it is irrelevant to be the referrer of the current web page or website. A genuine request will indicate that the source of the referral must come from valid resources.
- **False Proxy Header:** The existence of a public proxy that can be utilized freely causes the service to be misused by attackers to execute HTTP DDoS attacks. Besides, there is no restriction to adopt the public proxy which further increases attackers' capability to launch attacks without being detected. HTTP DDoS attacks launched through a public proxy deliver inconsistent proxy

headers that pose difficulties in recognizing whether the source of a request originates from proxy or directly from the client as well as in detecting whether the source request was legitimate or malicious.

- **Spoof IP Address:** An attacker utilizes a spoof IP address to execute HTTP DDoS attack to conceal the activity of overwhelming a web server. The analysis conducted in Section 9.6 indicates that the Wreckuests.py attack script adopted several machines and used public IP addresses with an attachment of proxy headers to make the source request to appear as if they originate from a proxy. The execution of this attack script provides a significant finding with regards to HTTP DDoS attack strategy discussed earlier in Section 4.

12. Attack Category Associated with Real HTTP DDoS

The attack script adopted in this research is well-known and has been utilized by many prior studies. However, none of the past studies perform any mapping of the attack scripts in terms of showing the linkage between the type of the attack script and its attack strategy. Thus, based on the attack patterns executed, this study suggests that the attack scripts belong to five categories: web proxies, constant, server load, main page attack and random attack. Table 9 presents details regarding the dataset mapping with its attack strategy.

Table 9. Attack script mapping with existing attack strategy.

No	Attack Strategy	Attack Script Name
1.	Web Proxies	Hibernet.py UFONet.py
2.	Spoofing	Wreckuests.py
3.	Server Load	Chihulk, High Orbit Ion Canon (HOIC), Golden Eye, BlackHorizon
4.	Main Page Attack	
5.	Random Attack	

13. Conclusions

In this paper, an analysis of forged request headers created by HTTP DDoS was conducted. It is important to note that the request header is not compulsory to be presented which enables cyber intruders to generate forged headers that are identical to legitimate HTTP request. This research adopted eight attack scripts which have been utilized and executed separately to obtain precise information regarding the attack patterns. The attacks were executed in both the internet and external networks to observe the attack behavior in launching the attacks in these networks. A critical analysis was conducted, and the results show that there are similarities and differences between false request headers generated by various attack scripts in both internal and external networks. The results thus indicate that manipulation of request headers was constituted in various ways by HTTP DDoS to make the HTTP request appear authentic. Apart from that, the request headers require a major improvement particularly in the area of security to prevent the header from being manipulated by attackers. The reinforcement of the security for an HTTP protocol will ensure that HTTP DDoS attacks can be easily detected in future.

14. Future Work

Future research is recommended to include an execution of HTTP DDoS attacks in HTTP version 2 to observe the attack strategy in the manipulation of request headers. HTTP version 2 has a different structure compared to HTTP version 1.1. Hence, further exploration investigating an execution of HTTP DDoS in HTTP version 2 to analyze the request header structure which can possibly be forged by HTTP DDoS attacks. As technology evolves, the approach for a web browser communicating with a web server also changes. Currently, most of the web browsers utilize HTTP version 1.1 whereby 90% of web servers are not migrating to HTTP version 2 [55] as the migration is time consuming.

The structures of HTTP version 2 still support basic features existed in HTTP 1.1 [56]. [57] also noted that the HTTP version 2 still employs several request header components which are used by HTTP version 1.1 such as user-agent, accept and accept-language as highlighted in Table 10.

Table 10. HTTP 2 GET headers.

Request 1 # Total Bytes 220	Request 2 # Total Byte 230
: Authority: www.akamai.com	: Authority: www.akamai.com
: Method: GET	: Method: GET
: Path:/	: Path:/style.css
: scheme: https	: scheme: https
accept: text/html,application/xhtml+xml	accept: text/html,application/xhtml+xml
accept-languange:en-US,en;q=0.8	accept-languange:en-US,en;q=0.8
cookie:last_page=286A7F3DE	**cookie:last_page=*398AB8E8F**
upgrade-insecure-request:1	upgrade-insecure-request:1
user-agent:mozilla http2	user-agent:mozilla http2

* Those in bold mark the difference.

Ludin and Garza [57] explained that only unique bytes were sent to a web server unlike HTTP 1.1 whereby the entire bytes are delivered to a web server. Table 10 illustrates the explanation pertaining to unique byte where the second request only sends the unique byte, which is only 10 bytes.

15. Possible Solutions and Open Issues

The outcome of this study indicated in Section 9 successfully reveals eight forged request headers constituted by HTTP DDoS attacks which leads to possible solutions for future researchers to further explore the various ways DDoS attacks are executed at the application layer. Among the possible areas for future research include:

(a) Development of a web browser without adopting secure socket layers to encrypt the GET header and its value to prevent header manipulation.
(b) Analysis of other devices acted as botnets which come from IoT devices to observe the forged GET header patterns.
(c) Execution of HTTP DDoS through HTTPS protocol and HTTP version 2 to explore the possible GET header manipulation.
(d) Development of new standards to reject HTTP request originating from a non-browser as done by HTTP DDoS adopted attack scripts to generate substantial requests.

Future researchers should also take note that vulnerabilities at the application layer not only lead to the occurrence of several types of attacks, but the shortfalls can also be utilized to design a defense system as discussed in Section 5. Future research should also take precautionary measures as there may be possible open issues that require further attention when providing solutions in detecting HTTP DDoS, including:

(a) The appearance of HTTP DDoS as the service increases the occurrence of the attack and becomes simpler to be executed.
(b) The existence of public proxies scattered around the world that provides opportunity for cyber intruders to utilize the service to launch HTTP DDoS attacks. The usage of proxies to execute HTTP DDoS increases the complexity to trace the source of the request. Besides that, the usage of proxies does not require any authentication which allows both genuine users and attackers to adopt the same proxy.
(c) A web server accepts any requestor without performing an inspection of the platform's source such as whether it comes through scripting, tools or web browsers. A web server is supposed to

be more selective when accepting eligible clients to be connected to a web server to gain access to the web server's content.

(d) The structure of the HTTP protocol allows the request headers and its data to be edited which leads to manipulation of request headers. A restriction should be applied to allow only specific request headers to prevent any header manipulation executed by attackers.

(e) The request header presented in HTTP request transaction should take note of the requestor information such as the web browser's name, language, operating system, etc. However, these headers are not compulsory to appear in each HTTP request transaction. Due to this circumstance, attackers are only required to generate an HTTP request without having any request header to overwhelm a web server. Besides, attackers often adopt request headers to make the header to appear authentic which lead to further complication in detecting both the DDoS and the authenticity of the request at the application layer.

(f) The existence of attack tools and scripts available on the Internet allows an HTTP DDoS to be easily executed. The appearance to the attack script provides an opportunity for researchers in conducting further research. However, the same attack script could also be adopted by an attacker as a medium to launch an attack.

(g) The proxy provider adopted a distinct name to indicate that a HTTP request originates from a proxy. The usage of a different name for a proxy header leads to confusion. Besides, the same proxy is likely to be adopted by both an authentic client and an attacker to access the same web server.

(h) A high-speed Internet delivers an advantage to users to browse online contents. However, the Internet speed provides a significant impact to substantial amount of HTTP requests that can be generated through a single requestor within a specific time period.

(i) The attack scripts adopted in this research were publicly available and are compulsory to be executed in the real network. A researcher requires a sound technical skill to configure those devices to simulate the attack in the Local Area Network (LAN) or Wide Area Network (WAN). The impact of an attack executed in the WAN network requires a higher attention as the Internet Service Provider (ISP) has possibly blocked the traffic therefore jeopardizing the research results.

(j) The development of a defense system which utilizes either signature-based, anomaly-based and hybrid detection methods must be selected with precaution as each detection method has its own strengths and limitations.

Author Contributions: Writing—original draft, Writing-review & editing, A.G.J. and S.A.I.; Investigation, S.A.I., N.K. and O.M.Y.; Methodology, M.S.A. and A.A.; Validation, M.S.A.; Formal analysis, N.K., A.A. and O.M.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research paper took place and received financial support from Universiti Teknologi Malaysia (UTM). Financial support number UTMER-Q.K130000.2656.17J67.

Conflicts of Interest: The authors declare that there is no conflict of interests regarding the publication of this paper.

References

1. Behal, S.; Kumar, K. Trends in validation of DDoS research. In *International Conference on Computational Modeling and Security (CMS 2016)*; Elsevier: Amsterdam, The Netherlands, 2016; Volume 85, pp. 7–15.
2. Jazi, H.H.; Gonzalez, H.; Stakhanova, N.; Ghorbani, A.A. Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling. *Comput. Netw.* **2017**, *121*, 25–36. [CrossRef]
3. Singh, K.; Singh, P.; Kumar, K. Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges. *Comput. Secur.* **2017**, *65*, 344–372. [CrossRef]
4. Behal, S.; Kumar, K. Detection of DDoS attacks and flash events using novel information theory metrics. *Comput. Netw.* **2017**, *116*, 96–110. [CrossRef]

5. Dhanapal, A.; Nithyanandam, P. An effective mechanism to regenerate HTTP flooding DDoS attack using real time data set. In Proceedings of the 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), Kannur, India, 6–7 July 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 570–575.
6. Singh, K.; Singh, P.; Kumar, K. User behavior analytics-based classification of application layer HTTP-GET flood attacks. *J. Netw. Comput. Appl.* **2018**, *112*, 97–114. [CrossRef]
7. Prasad, K.M.; Reddy, A.R.M.; Rao, K.V. BIFAD: Bio-Inspired Anomaly Based HTTP-Flood Attack Detection. *Wirel. Pers. Commun.* **2017**, *97*, 281–308. [CrossRef]
8. Zargar, S.T.; Joshi, J.; Tipper, D. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Commun. Surv. Tutor.* **2013**, *15*, 2046–2069. [CrossRef]
9. Kumar, V.; Kumar, K. Classification of DDoS attack tools and its handling techniques and strategy at application layer. In Proceedings of the 2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Fall), Bareilly, India, 30 September–1 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.
10. Osanaiye, O.; Choo, K.-K.R.; Dlodlo, M. Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. *J. Netw. Comput. Appl.* **2016**, *67*, 147–165. [CrossRef]
11. Ye, J.; Cheng, X.; Zhu, J.; Feng, L.; Song, L. A DDoS Attack Detection Method Based on SVM in Software Defined Network. *Secur. Commun. Netw.* **2018**, *2018*, 1–8. [CrossRef]
12. Gu, Y.; Wang, Y.; Yang, Z.; Xiong, F.; Gao, Y. Multiple-Features-Based Semisupervised Clustering DDoS Detection Method. *Math. Probl. Eng.* **2017**, *2017*, 1–10. [CrossRef]
13. Cheng, J.; Li, M.; Tang, X.; Sheng, V.S.; Liu, Y.; Guo, W. Flow Correlation Degree Optimization Driven Random Forest for Detecting DDoS Attacks in Cloud Computing. *Secur. Commun. Netw.* **2018**, *2018*, 1–14. [CrossRef]
14. Myint, O.M.; Kamolphiwong, S.; Kamolphiwong, T.; Vasupongayya, S. Advanced Support Vector Machine-(ASVM-) Based Detection for Distributed Denial of Service (DDoS) Attack on Software Defined Networking (SDN). *J. Comput. Netw. Commun.* **2019**, *2019*, 1–12. [CrossRef]
15. Cepheli, O.; Buyukcorak, S.; Karabulut Kurt, G. Hybrid Intrusion Detection System for DDoS Attacks. *J. Electr. Comput. Eng.* **2016**, *2016*, 1–8. [CrossRef]
16. Suroto, S. A review of defense against slow HTTP attack. *JOIV Int. J. Inform. Vis.* **2017**, *1*, 127–134. [CrossRef]
17. Yevsieieva, O.; Helalat, S.M. Analysis of the impact of the slow HTTP DOS and DDOS attacks on the cloud environment. In Proceedings of the 2017 4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 10–13 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 519–523.
18. Nithyanandam, P.; Dhanapal, A. The Slow HTTP Distributed Denial of Service Attack Detection in Cloud. *Scalable Comput. Pract. Exp.* **2019**, *20*, 285–298.
19. Subramanian, K.; Gunasekaran, P.; Selvaraj, M. Two Layer Defending Mechanism against DDoS Attacks. *Int. Arab J. Inf. Technol. (IAJIT)* **2015**, *12*, 317–324.
20. Hoque, N.; Bhattacharyya, D.K.; Kalita, J.K. Botnet in DDoS Attacks: Trends and Challenges. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2242–2270. [CrossRef]
21. Beitollahi, H.; Deconinck, G. ConnectionScore: A statistical technique to resist application-layer DDoS attacks. *J. Ambient Intell. Humaniz. Comput.* **2013**, *5*, 425–442. [CrossRef]
22. Wang, J.; Yang, X.; Zhang, M.; Long, K.; Xu, J. HTTP-SoLDiER: An HTTP-flooding attack detection scheme with the large deviation principle. *Sci. China Inf. Sci.* **2014**, *57*, 1–15. [CrossRef]
23. Zhao, Y.; Zhang, W.; Feng, Y.; Yu, B. A Classification Detection Algorithm Based on Joint Entropy Vector against Application-Layer DDoS Attack. *Secur. Commun. Netw.* **2018**, *2018*, 1–8. [CrossRef]
24. Niakanlahiji, A.; Chu, B.-T.; Al-Shaer, E. PhishMon: A machine learning framework for detecting phishing webpages. In Proceedings of the 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), Miami, FL, USA, 9–11 November 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 220–225.
25. Gahagan, I.V.J.; Bhansali, D.; Gratian, M.; Cukier, M. A comprehensive evaluation of http header features for detecting malicious websites. In Proceedings of the 2019 15th European Dependable Computing Conference (EDCC), Naples, Italy, 17–20 September 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 75–82.

26. Tyson, G.; Huang, S.; Cuadrado, F.; Castro, I.; Perta, V.C.; Sathiaseelan, A.; Uhlig, S. Exploring http header manipulation in-the-wild. In Proceedings of the 2017 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 451–458.
27. Mansoori, M.; Hirose, Y.; Welch, I.; Choo, K.-K.R. Empirical analysis of impact of HTTP referer on malicious website behaviour and delivery. In Proceedings of the 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), Crans-Montana, Switzerland, 23–25 March 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 941–948.
28. Xu, F.; Pan, H.; Cao, Z.; Li, Z.; Xiong, G.; Guan, Y.; Yiu, S.-M. Identifying malware with HTTP content type inconsistency via header-payload comparison. In Proceedings of the 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC), San Diego, CA, USA, 10–12 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–7.
29. Liu, S.; Wang, J.; Wang, H.; Wang, H.; Liu, Y. WRT: Constructing users' web request trees from http header logs. In Proceedings of the ICC 2019 International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–7.
30. La, V.H.; Fuentes, R.; Cavalli, A.R. Network monitoring using mmt: An application based on the user-agent field in http headers. In Proceedings of the 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), Crans-Montana, Switzerland, 23–25 March 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 147–154.
31. Niu, W.; Li, T.; Zhang, X.; Hu, T.; Jiang, T.; Wu, H. Using XGBoost to Discover Infected Hosts Based on HTTP Traffic. *Secur. Commun. Netw.* **2019**, *2019*. [CrossRef]
32. Saleh, M.A.; Manaf, A.A. A Novel Protective Framework for Defeating HTTP-Based Denial of Service and Distributed Denial of Service Attacks. *Sci. World J.* **2015**, *2015*, 238230. [CrossRef] [PubMed]
33. Yadav, S.; Selvakumar, S. Detection of application layer DDoS attack by modeling user behavior using logistic regression. In Proceedings of the 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), Noida, India, 2–4 September 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1–6.
34. Aceto, G.; Pescape, A. Internet censorship detection: A survey. *Comput. Netw.* **2015**, *83*, 381–421. [CrossRef]
35. Sree, T.R.; Bhanu, S.M.S. HADM: Detection of HTTP GET flooding attacks by using Analytical hierarchical process and Dempster-Shafer theory with MapReduce. *Secur. Commun. Netw.* **2016**, *9*, 4341–4357. [CrossRef]
36. Bravo, S.; Mauricio, D. DDoS attack detection mechanism in the application layer using user features. In Proceedings of the 2018 International Conference on Information and Computer Technologies (ICICT), DeKalb, IL, USA, 23–25 March 2018; pp. 97–100.
37. Sreeram, I.; Vuppala, V.P.K. HTTP flood attack detection in application layer using machine learning metrics and bio inspired bat algorithm. *Appl. Comput. Inform.* **2017**. [CrossRef]
38. Liao, Q.; Li, H.; Kang, S.; Liu, C. Application layer DDoS attack detection using cluster with label based on sparse vector decomposition and rhythm matching. *Secur. Commun. Netw.* **2015**, *8*, 3111–3120. [CrossRef]
39. Dhanapal, A.; Nithyanandam, P. An OpenStack based cloud testbed framework for evaluating HTTP flooding attacks. *Wirel. Netw.* **2019**. [CrossRef]
40. Rahman, R.; Tomar, D.S.; Jijin, A.V.J. Application Layer DDOS Attack Detection Using Hybrid Machine Learning Approach. *Int. J. Secur. Appl.* **2017**, *11*, 85–96. [CrossRef]
41. Ghafar, A.J.; Shahidan, M.A.; Adli, S. Review of Recent Detection Methods for HTTP DDoS Attack. *J. Comput. Netw. Commun.* **2019**, *2019*, 1–10.
42. Idhammad, M.; Afdel, K.; Belouch, M. Detection System of HTTP DDoS Attacks in a Cloud Environment Based on Information Theoretic Entropy and Random Forest. *Secur. Commun. Netw.* **2018**, *2018*, 1–13. [CrossRef]
43. Singh, K.J.; Thongam, K.; De, T. Entropy-Based Application Layer DDoS Attack Detection Using Artificial Neural Networks. *Entropy* **2016**, *18*, 350. [CrossRef]
44. Shiaeles, S.N.; Papadaki, M. FHSD: An Improved IP Spoof Detection Method for Web DDoS Attacks. *Comput. J.* **2014**, *58*, 892–903. [CrossRef]
45. Hameed, S.; Ali, U. HADEC: Hadoop-based live DDoS detection framework. *EURASIP J. Inf. Secur.* **2018**, *2018*, 11. [CrossRef]
46. Zhang, Y.; Mekky, H.; Zhang, Z.L.; Torres, R.; Lee, S.J.; Tongaonkar, A.; Mellia, M. Detecting malicious activities with user-agent-based profiles. *Int. J. Netw. Manag.* **2015**, *25*, 306–319. [CrossRef]

47. Gou, G.; Bai, Q.; Xiong, G.; Li, Z. Discovering abnormal behaviors via HTTP header fields measurement. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e3926. [CrossRef]
48. Fielding, R.T.; Reschke, J.F. Message Syntax and Routing. IETF RFC 7230. June 2014. Available online: https://tools.ietf.org/pdf/rfc7230 (accessed on 27 March 2019).
49. Reid, F. 4—HTTP: Communicating with Web Servers. In *Network Programming in NET*; Reid, F., Ed.; Digital Press: Burlington, NJ, USA, 2004; pp. 87–130.
50. Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Berners-Lee, T. RFC 2068: Hypertext Transfer Protocol—HTTP/1.1, January 1997. Status: Proposed Standard 1997. Available online: https://tools.ietf.org/pdf/rfc2616 (accessed on 27 March 2019).
51. Fielding, R.; Reschke, J. Hypertext transfer protocol (HTTP/1.1): Semantics and Content. 2070-1721. 2014. Available online: https://tools.ietf.org/pdf/rfc7231 (accessed on 27 March 2019).
52. Dolnak, I. Implementation of referrer policy in order to control HTTP Referer header privacy. In Proceedings of the 2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA), Prešovský, Slovakia, 26–27 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–4.
53. Wall, D. Chapter 3—HTTP in PHP. In *Multi-Tier Application Programming with PHP*; Wall, D., Ed.; Morgan Kaufmann: San Francisco, CA, USA, 2004; pp. 21–43.
54. Petersson, A.; Nilsson, M. Forwarded HTTP Extension. 2014. Available online: https://tools.ietf.org/pdf/rfc7239 (accessed on 27 March 2019).
55. Adi, E.; Baig, Z.; Hingston, P. Stealthy Denial of Service (DoS) attack modelling and detection for HTTP/2 services. *J. Netw. Comput. Appl.* **2017**, *91*, 1–13. [CrossRef]
56. Tripathi, N.; Hubballi, N. Slow rate denial of service attacks against HTTP/2 and detection. *Comput. Secur.* **2018**, *72*, 255–272. [CrossRef]
57. Ludin, S.; Garza, J. *Learning HTTP/2: A Practical Guide for Beginners*; O'Reilly Media, Inc.: Newton, MA, USA, 2017.