How Secure are Web Servers? An Empirical Study of Slow HTTP DoS Attacks and Detection

Nikhil Tripathi, Neminath Hubballi
Discipline of CSE, School of Engineering
Indian Institute of Technology Indore
{phd1401101002, neminath}@iiti.ac.in

Yogendra Singh
Computer Center
Indian Institute of Technology Indore
yogendras@iiti.ac.in

***Abstract*—Slow HTTP Denial of Service (DoS) is an application layer DoS attack in which large number of incomplete HTTP requests are sent. If number of such open connections in the server exhaust a preset threshold, server does not accept any new connections thus creating DoS. In this paper we make twofold contributions. We do an empirical study on different HTTP servers for their vulnerability against slow HTTP DoS attacks. Subsequently we propose a method to detect Slow HTTP Dos attack. The proposed detection system is an anomaly detection system which measures the Hellinger distance between two probability distributions generated in training and testing phases. In the training phase it creates a normal profile as a probability distribution comprising of complete and incomplete HTTP requests. In case of Slow HTTP attack the proportion of incomplete messages is increased in the overall traffic and detection system leverages this for detection by generating another probability distribution and finding difference between two probability distributions. We experiment by collecting data from a real web server and report the detection performance of proposed detection system.**

***Index Terms*—Slow HTTP attack, Denial of Service, Probability Distribution, Hellinger Distance**

I. Introduction

Application Layer Denial of Service (DoS) attacks are increasingly becoming a serious threat to to the working of web servers. In these attacks, a malicious client exploits application specific vulnerabilities to cause either Reduction of Quality (RoQ) of service or simply, DoS. These DoS attacks send specially crafted messages to exploit the specific behavior of applications (for example how requests are processed) instead of generating a huge flood of traffic to overwhelm the network. Causing DoS using these methods is much easier compared to traditional DoS attacks [33] as these attacks require to generate fewer number of messages. Low Rate HTTP DoS attack [19] is an example of Application Layer DoS attack. In this attack, a malicious client sends a reasonably large number of HTTP requests intelligently to occupy all available HTTP connections that are permitted on a web server. As a result, the server stops processing further requests sent by other genuine clients, resulting in DoS. Several previous works [17], [16], [20], [28], [18], [32] have studied the effects of Low Rate HTTP attacks and proposed countermeasures for this. However recently, a relatively newer class of Application Layer DoS attack known as Slow HTTP DoS attack has been discovered. This attak aims at exhausting queue memory of at the sevrer by keeping enough number of incomplete requests and prolonging the connection lifetime. To launch the attack, malicious client creates multiple connections with the web server and sends incomplete HTTP requests from each of these connections. Since these connections interact with server very slowly, the server stores them in a connection queue till these connections are completely served. Once all the available space in this queue is occupied, no legitimate connections are entertained by server, thereby, causing DoS attack. Since these attacks are latest discovery, it has received very limited attention of researchers.

There are several versions of Slow HTTP DoS attacks. In this paper, we focus on two most popular versions of Slow HTTP DoS attacks - *Slow Header Attack* and *Slow Message Body Attack*. We present a detailed study of these attacks by analyzing behaviour of different types of web servers and live websites[1]. We also describe a technique to detect these attacks. In particular we make following specific contributions in this paper.

- We study the impact of Slow HTTP DoS attacks on widely used web servers and report what type of attack is effective in what type of server.
- We perform an empirical study of Slow HTTP DoS attacks against a large number of websites falling into 4 categories and present worrying results. These web servers are hosted in different parts of the globe.
- Based on our empirical study we conclude that Slow HTTP DoS attacks are a real threat and hence believe that detecting these attacks is important. We subsequently propose a detection scheme to detect these attacks. The proposed detection system is an anomaly detection system which measures the Hellinger Distance between two probability distributions generated in training and testing phases.
- We experiment with varied rate of Slow HTTP DoS attack and show the detection performance of our proposed system.

Rest of this paper is organized as follows. In section II, we describe two types of Slow HTTP DoS attacks. In Section III we present an empirical study of Slow HTTP DoS attacks.

[1]These are our partner websites and are tested as part of vulnerability assessment

978-1-5090-0990-9/16 $31.00 © 2016 IEEE
DOI 10.1109/ARES.2016.20

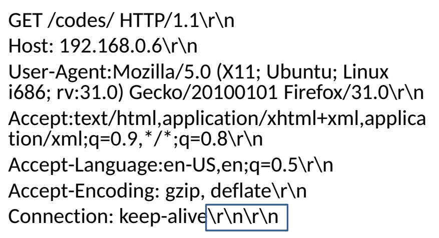
We present a detection system to detect Slow HTTP Attacks in Section IV. Experimental results are presented in Section V. Section VI presents related work followed by conclusion in Section VII.

II. Slow HTTP DoS Attacks

Both Low Rate HTTP DoS and Slow HTTP DoS attacks are aimed at creating denial of service. In Low Rate HTTP DoS attack complete HTTP requests are sent with the intention of being served. However in case of Slow HTTP DoS attack incomplete HTTP requests are sent with bogus headers with the intention of forcing server to wait for complete HTTP requests. Though both these attacks are intended to consume available connection queue space, Slow HTTP DoS attacks are more effective as compared to Low Rate HTTP DoS attacks. This is due to the fact that after serving a request, the amount of time for which a web server waits[2] before closing the connection is lesser than the amount of time it waits to receive a complete HTTP request. For instance, Apache web server after serving a request, waits for 26 seconds before closing the connection. On the other hand, it waits for around 300 seconds to receive a complete HTTP request. As a result, connections with partial HTTP requests are able to occupy the connection queue space for a longer time, thereby making Slow HTTP DoS attack more effective. In this section, we discuss two popular Slow HTTP DoS attacks. RFC 2616 [11] mentions different types of methods to be performed on the resource identified by the Request-URI. In this paper, we focus only on *GET* and *POST* methods[3]. *GET* method is used to retrieve data, e.g. index page, identified by Request-URI from a web server while *POST* method is used to send some data to the web server, e.g. data sent during a form submission. By exploiting the way these two methods behave, we can create slow HTTP DoS attacks as described below.

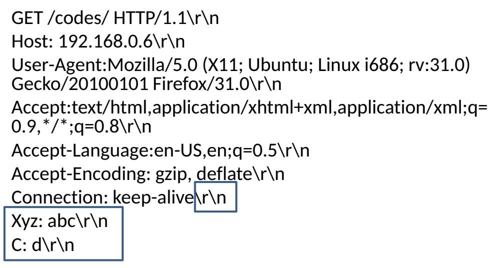
A. Slow Header Attack

This attack exploits the fact that HTTP protocol by its design, requires GET headers to be completely received before they are processed [11]. If an incomplete HTTP GET header is received, the server assumes that the client is operating from a slow Internet connection and thus keeps its resources busy waiting for the rest of the header. On the other hand, the malicious client never sends complete HTTP header. Instead, it just feeds bogus HTTP header fields to server at regular time intervals so that server resets the expiry timer every time it receives parts of header and thus, maintains the connection alive for a long time. According to RFC 2616, "$\backslash r\backslash n$" denotes the end of a line in HTTP header while "$\backslash r\backslash n\backslash r\backslash n$" denotes the end of a compelte HTTP header and beginning of message body. So by not transmitting "$\backslash r\backslash n\backslash r\backslash n$" in HTTP GET headers and sending bogus header fields instead, the malicious client can craft the desired incomplete HTTP GET requests. Figure 1 and Figure 2 shows such complete HTTP GET header and incomplete HTTP GET header with bogus header fields respectively. From Figure 2, we can also notice the absence of "$\backslash r\backslash n\backslash r\backslash n$" sequence in case of incomplete HTTP GET header. The malicious client creates enough connections to web server and sends incomplete HTTP GET requests from each of them to fill the connection queue of the web server so that legitimate clients can not connect to the server. Tools like Slowloris [29] and SlowHTTPTest have modules that can be used to generate this type of attack.



```
GET /codes/ HTTP/1.1\r\n
Host: 192.168.0.6\r\n
User-Agent:Mozilla/5.0 (X11; Ubuntu; Linux
i686; rv:31.0) Gecko/20100101 Firefox/31.0\r\n
Accept:text/html,application/xhtml+xml,applica
tion/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language:en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n\r\n
```

Fig. 1: Complete HTTP GET Request



```
GET /codes/ HTTP/1.1\r\n
Host: 192.168.0.6\r\n
User-Agent:Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0)
Gecko/20100101 Firefox/31.0\r\n
Accept:text/html,application/xhtml+xml,application/xml;q=
0.9,*/*;q=0.8\r\n
Accept-Language:en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
Xyz: abc\r\n
C: d\r\n
```

Fig. 2: Incomplete HTTP GET Request with Bogus Header Fields

B. Slow Message Body Attack

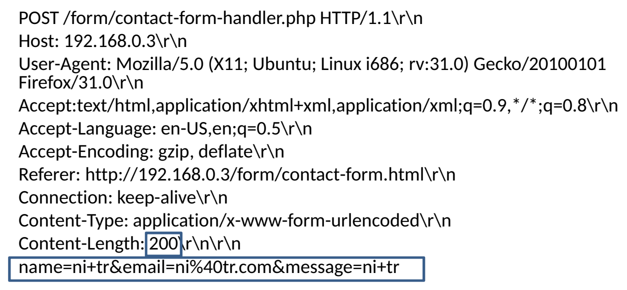
In this attack, the malicious client sends complete HTTP POST header however it sends the message body in very small chunks, thereby, forcing server to wait to receive complete message body. The *Content-Length* field of HTTP header indicates the size of message body in bytes. To generate this attack, the malicious client sends HTTP messages having higher *Content-Length* value than the actual size of message body. Figure 3. shows such HTTP POST message having *Content-Length* value of 200 while the actual message that is sent is of only 42 bytes (characters). When web server receives this message, it assumes that only a part of the message body is received and remaining part is still pending. Thus, server keeps its resources busy waiting for the rest of the message. The malicious client creates enough number of connections to web server and sends such HTTP messages from each of them to fill the connection queue of the web server, thereby, causing a DoS scenario. Tools like RUDY [27] and SlowHTTPTest have modules that can be used to create this type of attack.

III. Severity of Slow HTTP DoS Attacks

In this section, we report the impact of Slow HTTP attacks discussed in previous section against four most prominent web

[2]Here we assume persistent HTTP connections.

[3]Refer to [11] for details of other methods.



```
POST /form/contact-form-handler.php HTTP/1.1\r\n
Host: 192.168.0.3\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0) Gecko/20100101
Firefox/31.0\r\n
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Referer: http://192.168.0.3/form/contact-form.html\r\n
Connection: keep-alive\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 200\r\n\r\n
name=ni+tr&email=ni%40tr.com&message=ni+tr
```

Fig. 3: Larger Content-Length Value of 200 Bytes (in first box) than Actual Message Body of 42 Bytes (in second box) during Slow Message Body Attack

servers used in practice. We also report the outcome of these attacks against a large number of live web servers. For the sake of empirical study we chose 100 web servers hosted in different parts of the globe falling into four types.

A. *Testing in Testbed*

To evaluate the effectiveness of Slow HTTP DoS attacks against different web servers, we selected four servers - Apache, Microsoft IIS, Nginx and Lighttpd which are amongst most popular web servers according to the Netcraft's latest survey [31]. The versions of web servers we used for this study are Apache 2.4.18, IIS 7.5, Nginx 1.4.6 and Lighttpd 1.4.33. All of these servers are tested in their default configuration. Apache and IIS web servers are installed on a Windows 7 Home Premium machine having Intel Core 2 Duo processor with 4 GB of physical memory while NGinx and Lighttpd servers are installed on a Linux Mint machine having AMD Athlon X2 270 Dual core processor with 4 GB of physical memory. One of the clients running Ubuntu 14.04 having Intel Core 2 Duo processor with 1 GB of physical memory is designated as malicious client. This malicious client is configured with SlowHTTPTest framework. For each type of Slow HTTP DoS attacks, the observations on each of the four web servers are recorded. We developed a small website with 10 pages and few images and couple of forms with 5 text fields for testing purposes. Findings of Slow HTTP attack with GET and POST methods are summarized in Table I.

TABLE I: Slow HTTP DoS Vulnerability Assessment for Different Web Servers

Web Server	**Slow Header**	**Slow Message Body**
Apache	Vulnerable	Vulnerable
IIS	Not Vulnerable	Vulnerable
Nginx	Not Vulnerable	Not Vulnerable
Lighttpd	Vulnerable	Vulnerable

I. Apache Web Server: For GET request we observed that Apache web server closes a connection if it does not receive any data from that connection for 300 seconds. However, if malicious client continues to send bogus header fields at regular intervals, the server keeps waiting to receive complete HTTP GET Request for up to 990 seconds. After this timeout, the server responds with a 400 *Bad Request* message and finally closes the connection. This large waiting time of 990 seconds in the hope of receiving complete HTTP GET request makes Apache web server vulnerable to Slow Header attack. In case of Slow Message Body attack, we observed that if malicious client continues to send chunks of message body at regular intervals, Apache web server waits for indefinite amount of time to receive complete message body. As a result, Apache web server is vulnerable to Slow Message Body attack also.

II. IIS Web Server: In case of Microsoft's IIS web server, we observed that it closes a connection if it does not receive any data from that connection for 130 seconds. Even if malicious client continues to send bogus header fields at regular intervals, the server waits for just around 130 seconds to receive complete HTTP GET Request and then closes the connection. Also, IIS does not consider incomplete requests as writable until complete header is received. Such connections are not transferred to server's internal processing queue. As a result, IIS web server is not vulnerable to Slow Header attack. In case of Slow Message Body attack, due to presence of complete header (but incomplete message body), the requests are passed to server's internal processing queue. We also observed that if malicious client continues to send chunks of message body at regular intervals, IIS web server keeps waiting for indefinite amount of time to receive complete message body. As a result, IIS is vulnerable to Slow Message Body attack.

III. Nginx Web Server: In case of Nginx web server, we observed that it closes a connection if it does not receive complete HTTP header within 60 seconds. In case of Slow Message Body attack, we observed that if malicious client continues to send chunks of message body at regular intervals, the web server keeps waiting for indefinite amount of time to receive complete message body. However, when we tested the server by generating Slow Message Body attack using 1500 connections at the rate of 200 connections/second, the web server was able to handle and service all requests. The reason is Nginx uses a highly scalabale event-driven asynchronous architecture that makes use of small but predictable amounts of memory under load. According to Nginx technical specfications guide [26], it can handle upto 1 million concurrent connections. Also, Nginx handles requests using a single (or at least, very few) thread, thus consumes very less RAM dissimilar to process-based servers like Apache that requires a thread for each simultaneous connection. Due to these reasons, Nginx is not vulnerable to both Slow Header and Slow Message Body attack.

IV. Lighttpd Web Server: While analyzing the behavior of lighttpd web server, we observed that it closes a connection if it does not receive any data from that connection for around 60 seconds. However, if malicious client continues to send bogus header fields at regular intervals, the server keeps waiting to receive complete HTTP GET Request. This indefinite waiting to receive complete HTTP header makes lighttpd vulnerable to

Slow Header attack. In case of Slow Message Body attack, we observed that if malicious client continues to send chunks of message body at regular intervals, lighttpd web server keeps waiting for indefinite amount of time to receive complete message body. As a result, lighttpd web server is vulnerable to Slow Message Body attack too.

B. Empirical Evaluation of Slow HTTP Attacks

In order to assess the impact of Slow HTTP attacks we conducted a pilot study on 100 websites from our partners as part of an vulnerability assessment exercise. These websites are hosted on different types of web servers and are located in different parts of the world. For this pilot study we used only Slow HTTP GET attack as HTTP POST requires a custom message to be used (the number of fields and their types are different for each form). The websites chosen for this study are again carefully selected from four different categories. Owing to the privacy and security issues we name these categories as category -1 to category-4 without revealing their identity. We selected 25 websites from each of these four categories.

For the evaluation, we used two machines - one as malicious client to launch the attack and another for server availability of website under consideration during attack period. The malicious client was configured with SlowHTTPTest framework and the availability testing client was configured with one of the most popular benchmarking software, JMeter [14]. This software is an open source pure Java application designed by Apache itself for the purpose of load and performance testing. It simulates virtual users and generates web traffic towards a web server on which performance testing is going on. We connected these two clients to Internet using services of two different Internet Service Providers (ISP). We required different Internet connection for attacking and availability testing purpose because of possible presence of a Web Application Firewall (WAF) at server side that may blacklist an IP address from which it receive number of connections more than a particular threshold. Using SlowHTTPTest, we launched Slow Header attack on each of these 100 websites at a rate of 200 connections per second for a time period of 1 minute. Simultaneously, Jmeter was started on availability testing client to check the availability of these websites during attack period. We considered all those websites as vulnerable which suffered either from RoQ of service or DoS. After performing this pilot study, we obtained few worrying results. The most vulnerable class of websites was category-1 with 14 out of 25 websites vulnerable. Each of category-2 and category-3 had 8 out of 25 websites vulnerable. Category-4 was the least vulnerable with only 4 out of 25 websites vulnerable. Figures 4, 5, 6 and 7 show pie-charts for DoS vulnerability assessment of category-1, category-2, category-3 and category-4 respectively. In summary, we can notice that a large portion of web servers deployed in the wild are still vulnerable to Slow HTTP DoS attacks. Given the ease with which these attacks can be launched it is important to detect these attacks.

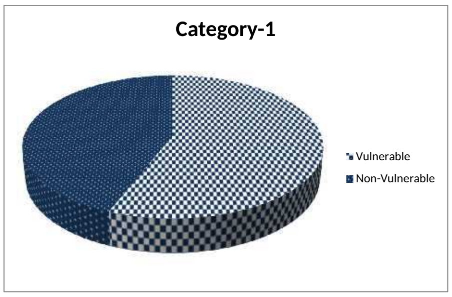


Fig. 4: Vulnerability Assessment of Category-1 Websites

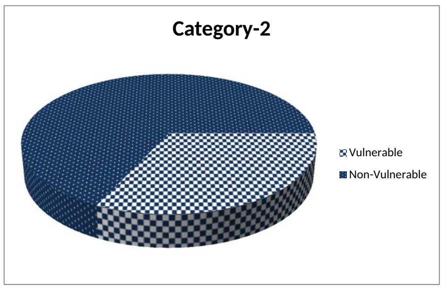


Fig. 5: Vulnerability Assessment of Category-2 Websites

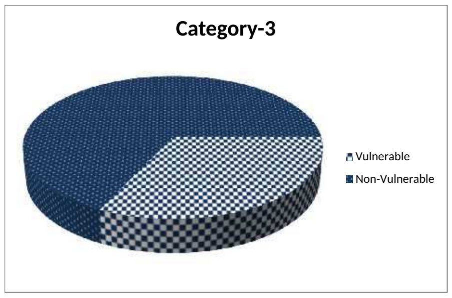


Fig. 6: Vulnerability Assessment of Category-3 Websites

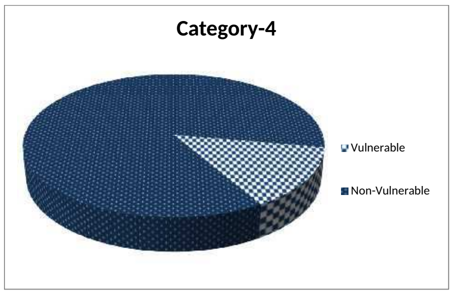


Fig. 7: Vulnerability Assessment of Category-4 Websites

IV. Detecting Slow HTTP DoS Attack

In this section, we describe a method to detect Slow HTTP Header and Slow HTTP Message Body DoS attacks. We describe a statistical abnormality measurement technique to detect these two attacks. In particular, we treat the normal web traffic as a probability distribution comprised of following four types of HTTP requests - Complete GET, Complete POST, Incomplete GET and Incomplete POST HTTP requests. In general a large portion of HTTP traffic consists of complete HTTP headers and complete HTTP message body. A very small portion of HTTP traffic arriving at the server will be comprised of incomplete requests and posts. However in case of Slow HTTP DoS attack this balance will be disturbed either due to increased number of incomplete HTTP GET requests in case of Slow Header attack or due to increased number of incomplete HTTP POST requests in case of Slow Message Body attack. We exploit this change in observation to detect these attacks. We use following methods to differentiate various request types.

- Complete/Incomplete HTTP GET Requests: If our detection system observes the presence of "$\backslash r\backslash n\backslash r\backslash n$" string in a HTTP GET request, it considers that request as complete HTTP GET request. Otherwise, request is considered as incomplete GET request.
- Complete/Incomplete HTTP POST Requests: To differentiate between complete and incomplete HTTP POST requests, our detection system compares the value present in *Content-Length* field of HTTP header and actual size of the message body. If both are equal, the request is considered as complete HTTP POST request. Otherwise, request is considered as incomplete POST request.

Like any other anomaly detection system our proposed detection method has two phases of operation as training phase and testing phase. In the training phase, we collect and create normal behavior profile of HTTP traffic (distribution graph) and in the testing phase, we compare the profile of current HTTP traffic with previously generated profile to detect slow HTTP DoS attacks. In particular, we use Hellinger Distance [7] between the training and testing probability distributions to detect the attacks.

A. Hellinger Distance

Given two probability distributions $\mathcal{P}$ and $\mathcal{Q}$, we use Hellinger distance to measure the difference between $\mathcal{P}$ and $\mathcal{Q}$. Here $\mathcal{P}$ and $\mathcal{Q}$ are N dimensional vectors with each component of the vector representing the probability of an attribute. The Hellinger distance is given by Equation 1. In Equation 1, P_i and Q_i denote the i^{th} components of vectors $\mathcal{P}$ and $\mathcal{Q}$.

$$d_H = (\frac{1}{2}\Sigma_{i=1}^{N}(\sqrt{P_i} - \sqrt{Q_i})^2)^{\frac{1}{2}} \tag{1}$$

The value of d_H will be always in between 0 and 1 with 0 representing perfect similarity and 1 representing maximum dissimilarity between $\mathcal{P}$ and $\mathcal{Q}$.

B. Adapting Hellinger Distance for Slow HTTP DoS attacks Detection

The proposed detection system has following two components.

- *Probabilistic Distribution of Training Data*: In order to create a profile of normal HTTP traffic, we observe and generate a distribution profile using four types of HTTP requests as discussed earlier over a period of n observation intervals each of duration ΔT. The profile generated in the training phase $\mathcal{P}$ has 4 attributes as below.
  $\mathcal{P} = \{P_{GETcomplete}, P_{POSTcomplete}, P_{GETincomplete}, P_{POSTincomplete}\}$ where $P_{GETcomplete}$, $P_{POSTcomplete}$, $P_{GETincomplete}$ and $P_{POSTincomplete}$ represent the probability of occurrence of complete GET, complete POST, incomplete GET and incomplete POST request respectively. Probability of a HTTP request of type P_i is estimated using Equation 2 where N_i denote the count of number of events of type i during the entire training period $(n * \Delta T)$ and N_{total} represent the count of events of all 4 types in the same duration.

$$P_i = \frac{N_i}{N_{total}} \tag{2}$$

- *Probabilistic Distribution of Testing Data*: Once the system is trained and $\mathcal{P}$ is generated, we use it to detect Slow HTTP DoS attacks from $(n+1)^{th}$ interval of duration ΔT. We generate a probability distribution $\mathcal{Q}$ every ΔT interval using Equation 2. In this case N_i represent the count of events of type i in the interval of consideration and N_{total} represent sum of all 4 types of events in the same interval. $\mathcal{Q}$ generated thus is compared with $\mathcal{P}$ using Hellinger distance. If the distance between the two distributions is greater than a threshold δ, $\mathcal{Q}$ is detected as anomaly otherwise it is considered as normal.

V. Experimental Results

In order to evaluate the proposed detection scheme, we acted in two different phases - training phase and testing phase. The training phase involved creation of legitimate HTTP traffic's profile. The testing phase involved comparison of current web traffic's profile and the profile created using legitimate HTTP traffic during training phase. We used two different sources for HTTP data collection, first by configuring a web server and simulating few virtual users on one of our departmental network and secondly by collecting real traces from our institution's public web server that hosts institute official website. In such a manner, we took both simulated and real HTTP traffic into account to evaluate our proposed detection scheme. In next few subsections, we describe the methods used to generate data for training and testing phases using simulated and real HTTP traffic and subsequently discuss the working of proposed detection scheme in detail.

TABLE II: Values for Uniform Delay Timer

Request	β (in seconds)	γ (in seconds)
HTTP GET Request-1	101	6
HTTP GET Request-2	101	5
HTTP GET Request-3	101	8
HTTP GET Request-4	101	6
HTTP GET Request-5	101	11
HTTP GET Request-6	101	5
HTTP GET Request-7	101	16
HTTP GET Request-8	101	21
HTTP POST Request	101	26

A. Data Collection for Training using Simulated HTTP Traffic

We created our first source of study by configuring a web server in one of departmental network. This machine was running Apache web server software v2.4.17 on Linux Mint operating system and having AMD Athlon X2 270 Dual core processor with 4 GB of physical memory. We designed a sample website having 7 web pages and 1 HTML form with 9 fields and hosted it on the web server. In the same network, we designated one of the client as virtual users simulator that generates legitimate HTTP traffic towards configured web server. This client was running Windows 7pro (service pack 1) having Intel Core2 Duo processor with 4 GB of physical memory. This client was equipped with Apache JMeter benchmarking tool to simulate 150 concurrent virtual users accessing the sample website. Each of the virtual user generated 8 HTTP GET and 1 HTTP POST request. All these virtual users were using non-persistent HTTP connections. As a result, once request from a connection was served, the client closed the connection without any delay. Also, these virtual users were kept in an uninterrupted loop so that they continued to send HTTP requests untill and unless they were manually terminated. JMeter provides various timer modules to integrate delay mechanisms while simulating virtual users. We used one such timer module called Uniform Random Timer. This timer generates time delays according to the Equation 3.

$$Delay = \alpha * \beta + \gamma \tag{3}$$

where α is a randomly generated value between 0 and 1 inclusively, β is the Random Delay Maximum and γ is the Constant Delay Offset. Table II shows the different values of these variables that we used for each of the HTTP requests. For simplicity, we kept value of $\beta = 101$ constant for all HTTP requests and in order to make these virtual users behave quite similar to real users, we chose values of γ according to the web page size. If size of the requested web page of the sample website is larger, a higher value of γ was chosen. For example, a real user, in general, needs more time to grasp contents of a webpage having more information as compared to a web page with very small information [34]. Also, a real user needs more time to fill a HTML form and so, we chose a higher value, $\gamma = 26$ for HTTP POST request as shown in Table II.

We collected 10 hours of simulated HTTP traffic generated by JMeter. We used the first 5 hours' traffic for training purpose i.e. to create distribution profile $\mathcal{P}$. We set the interval duration ΔT to 10 minutes for our experiments. Thus, there were a total of 30 such intervals from 5 hours of training data. We generated the probability distribution graph using these 30 interval data which represent the normal HTTP traffic. Figure 8 shows the distribution graph generated from this data. We can notice that probability of incomplete GET and incomplete POST requests is 0 because no incomplete requests were seen in training period. Moreover, we also notice that complete GET requests appeared in majority as compared to complete POST requests, thereby, having higher probability. This was mainly due to the large number of pages which have objects and other details and there was only one page having a form which uses POST method.

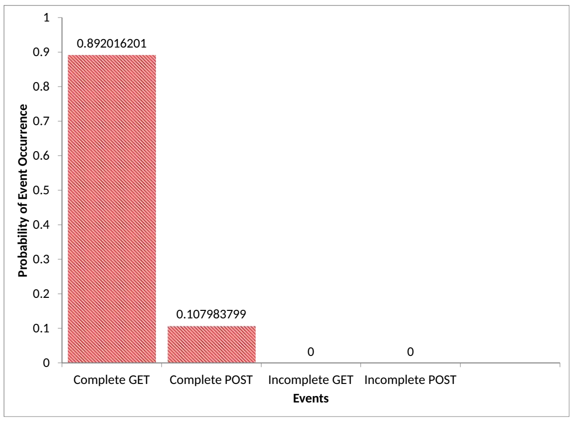


Fig. 8: Probability Distribution Generated from Simulated Traffic Training Data

B. Data Collection for Testing using Simulated HTTP Traffic

We used remaining five hours' normal HTTP traffic for testing purpose. Similar to training phase, we used intervals of $\Delta T = 10$ minutes in testing phase too. As a result, there were a total of 30 such intervals from these five hours' normal HTTP traffic. As mentioned previously, we measure the Hellinger distance between probability distributions of normal and testing interval. The minimum and maximum Hellinger distances between different intervals and normal interval are shown in Table III. In two time intervals of testing phase, we launched Slow Header attack and Slow Message Body attack using SlowHTTPTest framework at a rate of 1 connection per second. The Hellinger distances between these intervals and training profile are also shown in Table III with red colors. Since we launched each of these two attacks for only one interval, the minimum and maximum Hellinger distances are same. We can clearly notice that the estimated Hellinger distances in case of DoS attack scenarios is quite high as compared to the Hellinger distances estimated in case of normal web traffic. The sample probability distributions generated from testing normal interval, in case of Slow Header attack and Slow Message Body attack are shown in Figure 9,

TABLE III: Detection of Normal and Starvation Scenarios from Simulated HTTP Traffic

Type	Number of Intervals	Intervals Detected as Normal	Intervals Detected as Anomaly	Min/Max Hellinger Distance
Normal	28	28	0	0.0006/0.0118
Slow Header	1	0	1	0.3980
Slow Message Body	1	0	1	0.3971

Figure 10 and Figure 11 respectively. We can see that Figure 9 is more similar to the one generated in training phase and results in smaller distance value. By setting a threshold value of $\delta = 0.2$ we are able to detect both the intervals as anomaly.

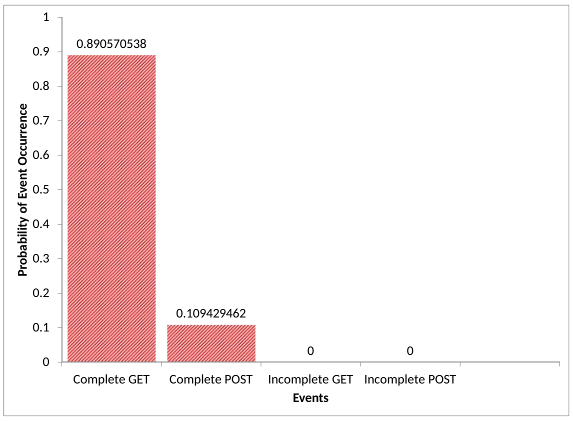


Fig. 9: Probability Distribution Generated from a Simulated Testing Normal Interval

C. *Data Collection for Training using Real HTTP Traffic*

We chose our institutional public web server that hosts institute official web site as our second source for training

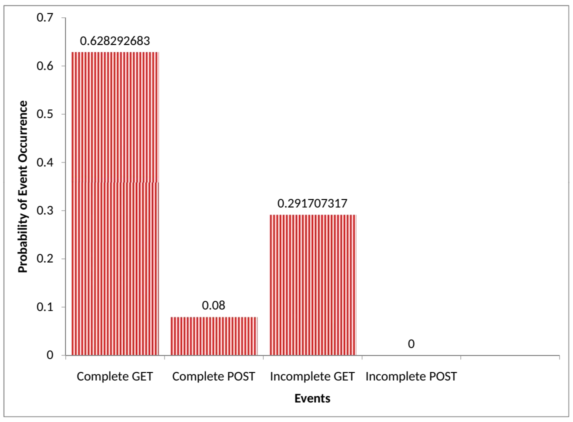


Fig. 10: Probability Distribution Generated from a Slow Header Attack Interval of Simulated HTTP Traffic

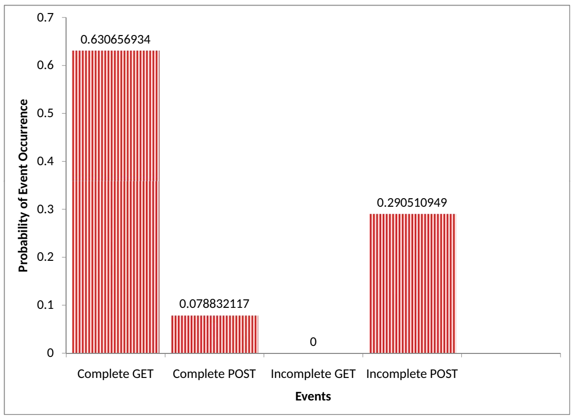


Fig. 11: Probability Distribution Generated from a Slow Message Body Attack Interval of Simulated HTTP Traffic

data. We collected traces from this web server so that we could obtain real HTTP traffic over Internet coming from clients all over the globe. This machine was running Apache web server software v2.2.22 on RHEL 6 having Intel Xeon CPU E5-26650 2.4GHz with 8 GB physical memory.

We collected 16 hours of real HTTP traffic from institution's public web server and used first 8 hours' traffic for training purpose. Similar to case of simulated HTTP traffic, we set the interval duration ΔT to 10 minutes and thus obtained a total of 48 training intervals. Figure 12. shows the distribution graph generated from this data. We can notice that, unlike simulated data this dataset has few incomplete GET and POST requests as well.

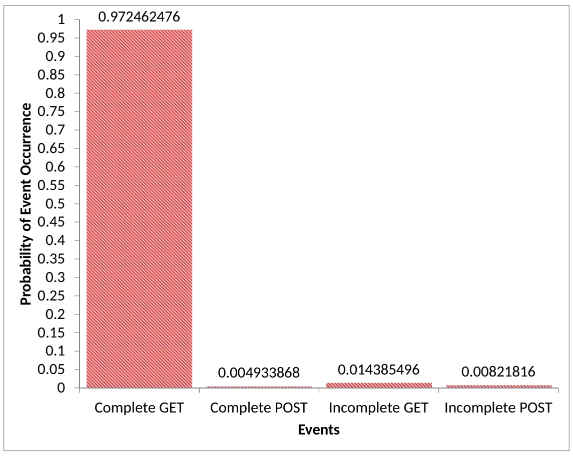


Fig. 12: Probability Distribution Generated from Real Traffic Training Data

D. *Data Collection for Testing using Real HTTP Traffic*

We used remaining eight hours' normal real HTTP traffic for testing purpose. Similar to training phase, we used intervals

TABLE IV: Detection of Normal and Starvation Scenarios from Testing a Real Traffic Normal Interval

Type	Number of Intetvals	Intervals Detected as Normal	Intervals Detected as Anomaly	Min/Max Hellinger Distance
Normal	46	46	0	0.0191/0.1273
Slow Header	1	0	1	0.2812
Slow Message Body	1	0	1	0.3562

of $\Delta T = 10$ minutes in testing phase too. As a result, there were a total of 48 such intervals from these eight hours' out of which in two intervals we launched Slow HTTP Header and Body attack using SlowHTTPTest framework at a rate of 1 connection per second. Thus there are total of 46 normal HTTP traffic intervals and 2 anomaly intervals. Subsequently we measured the Hellinger distance between probability distributions of these testing intervals. The estimated minimum and maximum Hellinger distances between different intervals and normal interval are shown in Table IV. The Hellinger distances between attack intervals and training profile are also shown in Table IV with red colors. Since we launched each of these two attacks for only one interval, the minimum and maximum Hellinger distances are same. We can clearly notice that the estimated Hellinger distances in case of DoS attack scenarios is quite high as compared to the Hellinger distances estimated in case of normal web traffic. Using real HTTP traffic, the sample probability distributions generated from testing normal interval, in case of Slow Header attack and Slow Message Body attack are shown in Figure 13, Figure 14 and Figure 15 respectively. We can also see that Figure 13 is more similar to the one generated in training phase and results in smaller distance value.

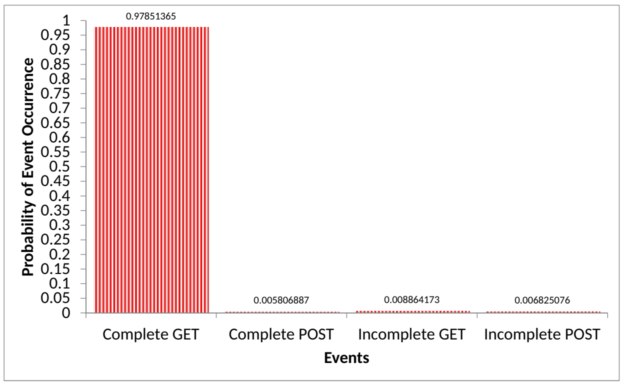


Fig. 13: Probability Distribution Generated from Testing a Real Traffic Normal Interval

It is worth noting that evading this detection is possible only when a malicious client generates HTTP requests such that distribution profile is similar to the one generated in training period. We noticed incomplete HTTP requests are not seen

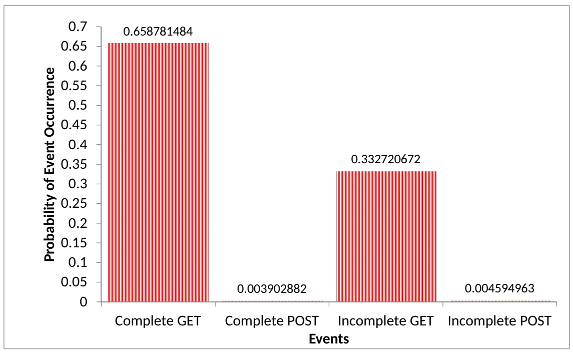


Fig. 14: Probability Distribution Generated from Testing a Slow Header Attack Interval of Real HTTP Traffic

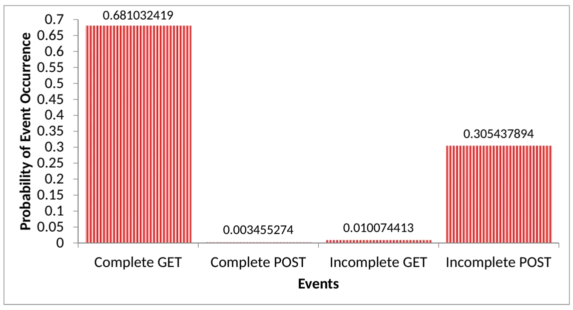


Fig. 15: Probability Distribution Generated from Testing a Slow Message Attack Interval of Real HTTP Traffic

or very rarely seen but in Slow HTTP DoS attacks, each attempt generates an incomplete HTTP request. These requests not only make the incomplete HTTP requests' probability go higher but also causes other probability of complete HTTP requests to reduce and this makes evading detection difficult.

VI. Related Work

The research community has invested considerable effort to detect Low Rate HTTP DoS attacks and thus, a huge amount of literature is available on this class of attack (see, e.g. Section II of [33]). Here, we summarize schemes that focus to combat Slow HTTP DoS attacks.

Various schemes have been proposed to detect anomalies present in web traffic. Anomaly-based detection schemes are more easily deployable than other complicated tools that are derived from machine learning as it simply looks at deviations from normal flows statistics [30], [4] while providing good detection rates [9]. Authors of [2] proposed a scheme that extracts traffic features like amount of time required to generate a HTTP request. Based on this feature, a comparison is made between sampling distribution generated from unknown traffic, potentially anomalous, and distribution generated from legitimate HTTP traffic to detect anomalies. Moreover, the authors assumed that for a request, composed by a single packet, duration of the request will be considered as 0. However, a malicious client can easily evade this scheme by sending an incomplete HTTP Request contained within a single packet without sending any bogus headers in subsequent packets. In such a way, malicious client still gets enough time (e.g. 300 seconds instead of 990 seconds in case of Apache web server) to launch Slow HTTP DoS attacks. In another work, Giralte et al. [12] used three types of analyzers which includes statistical analysis of HTTP flow, users' access behavior by the use of a graph to model the several paths of the web server as well as the different costs of navigating through the server and finally, the frequency of HTTP operations carried on the web server. However, the proposed technique can not detect *Distributed* Slow HTTP DoS attack if number of involved bots is large and each bot contributes to lower amount of attack traffic to evade detection. Moreover, this behavior can be easily simulated in order to avoid detection. The authors of [1], [24] presented a technique that tracks number of packets a web server receives in a given period of time. This feature is monitored in two subsequent temporal periods to detect normal or anomaly behavior. The weakness of this technique lies in the feature selection itself. Monitoring number of packets received at a web server can cause high false positive rate because a high burst of traffic can be due to scenarios like Flash Event [5] too. Moreover, these techniques can not detect Slow HTTP DDoS attacks [10].

Few patches [8], [21], [22], [23] are made available for apache server to mitigate Slow HTTP DoS attacks. However, these patches have limitation of blocking legitimate requests from being served. Moreover, authors of [25] showed that a web server configured with these modules, though, repulsed the attack; there was a noticeable delay in the server's response when attack was launched by 4 bots. Thus, it is possible that a more massive attack scenario can affect the server's performance.

In [3], authors proposed several neural models for the analysis of HTTP traffic to detect HTTP DoS attacks. With the help of experimental results, authors concluded that none of the applied neural models was able to differentiate normal from anomalous traffic. Authors of [6] proposed a lightweight method to detect DDoS attack by using traffic flow features. In particular, the authors used NOX platform [13] to facilitate handling of switch information. Also, the authors used Self Organizing Maps [15] for flow analysis to detect anomalies.

Table V. shows a comparison of our proposed detection technique with few previously proposed anomaly detection based schemes. From this comparison we conclude that the proposed detection system is better equipped to detect the Slow HTTP DoS attacks.

VII. Conclusion

In this paper, we presented a vulnerability assessment report of different web servers and some live websites against Slow HTTP DoS attacks. We presented results to show that Slow HTTP DoS attacks are still a threat to Internet despite of few known detection techniques. Subsequently we proposed a statistical abnormality measurement based detection system that measures Hellinger distance between two probability distribution generated during training phase and testing phase. This probability distribution comprises of complete and incomplete GET and POST requests. During attack period, the proportion of incomplete requests increases in the overall traffic that causes the corresponding probability distribution deviated from the normal probability distribution, resulting into anomaly. We evaluated the effectiveness of proposed detection technique by collecting simulated HTTP traffic generated in a LAN and real HTTP traces collected from a public web server too. Our experimental results show that proposed system detects Slow Header and Slow Message Body attacks with high accuracy during the attack period.

Acknowledgements

We thank all our partner entities who provided permission for vulnerability assessment on their website platforms. We also thank our Institute network administration team for their support in collecting traffic of web server.

References

[1] M. Aiello, E. Cambiaso, M. Mongelli, and G. Papaleo. An On-line Intrusion Detection Approach to Identify Low-rate DoS Attacks. In *2014 International Carnahan Conference on Security Technology (ICCST)*, pages 1–6, 2014.

[2] M. Aiello, E. Cambiaso, S. Scaglione, and G. Papaleo. A Similarity based Approach for Application DoS Attacks Detection. In *2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 430–435, 2013.

[3] D. Atienza, Á. Herrero, and E. Corchado. *International Joint Conference: CISIS'15 and ICEUTE'15*, chapter Neural Analysis of HTTP Traffic for Web Attack Detection, pages 201–212. 2015.

[4] T. Benmusa, D. J. Parish, and M. Sandford. Detecting and Classifying Delay Data Exceptions on Communication Networks Using Rule Based Algorithms. *International Journal of Communication Systems*, 18(2):159–177, 2005.

[5] S. Bhatia, G. Mohay, A. Tickle, and E. Ahmed. Parametric Differences Between a Real-world Distributed Denial-of-Service Attack and a Flash Event. In *2011 Sixth International Conference on Availability, Reliability and Security (ARES)*, pages 210–217, 2011.

[6] R. Braga, E. Mota, and A. Passito. Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, LCN '10, pages 408–415, 2010.

[7] X. Cao. Model Selection Based on Expected Squared Hellinger Distance, 2007. Ph.D Thesis, Colorado State University.

[8] Core. https://httpd.apache.org/docs/2.4/mod/core.html.

[9] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, and E. Meeuwissen. *Flow-based Detection of DNS Tunnels*. Springer, 2013.

[10] P. Farina, E. Cambiaso, G. Papaleo, and M. Aiello. Are Mobile Botnets a Possible Threat? The Case of SlowBot Net. *Computers & Security*, 58:268 – 283, 2016.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. (RFC 2616) Hypertext Transfer Protocol – HTTP/1.1, 1999.

[12] L. C. Giralte, C. Conde, I. M. D. Diego, and E. Cabello. Detecting Denial of Service by Modelling Web-Server Behaviour. *Computers & Electrical Engineering*, 39(7):2252–2262, 2013.

[13] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards An Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[14] JMeter. http://jmeter.apache.org/.

[15] T. Kohonen. The Self-Organizing Map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[16] G. Maciá-Fernández, J. E. Díaz-Verdejo, and P. García-Teodoro. Evaluation of A Low-Rate DoS Attack against Iterative Servers. *Computer Networks*, 51(4):1013–1030, 2007.

[17] G. Maciá-Fernández, J. E. Díaz-Verdejo, and P. García-Teodoro. Evaluation of A Low-Rate DoS Attack against Application Servers. *Computers & Security*, 27(7):335–354, 2008.

TABLE V: Comparison of Our Scheme with other Related Works

	Method	Traffic Data Collection	Can Slow HTTP *Distributed* DoS Attack be detected?
M. Aiello et al.[2]	Compare the time taken to complete a HTTP request in normal and attack scenario	Can be evaded by sending an incomplete HTTP Request contained within a single packet without sending any bogus headers in subsequent packets.	No
L. C. Giralte et al. [12]	Compares a real user's web access behaviour with web access behaviour of bots.	Can be bypassed by simulating real user's behaviour intelligently.	No
M. Aiello et al. [1], [24]	Compares number of packets received in a time period during normal and attack scenario.	High false positive rate	No
Proposed scheme	Compares number of complete and incomplete GET and POST requests received in a time period during normal and attack scenario.	High detection accuracy and low false positive rate.	Yes

[18] G. Maciá-Fernández, J. E. Díaz-Verdejo, and P. García-Teodoro. Mathematical Model for Low-Rate DoS Attacks against Application Servers. *IEEE Transactions on Information Forensics and Security*, 4(3):519–529, 2009.

[19] G. Maciá-Fernández, J. E. Díaz-Verdejo, P. García-Teodoro, and F. de Toro-Negro. LoRDAS: A Low-Rate DoS Attack against Application Servers. In *Critical Information Infrastructures Security*, pages 197–209. Springer Berlin Heidelberg, 2007.

[20] G. Maciá-Fernández, R. A. Rodríguez-Gómez, and J. E. Díaz-Verdejo. Defense Techniques for Low-Rate DoS Attacks against Application Servers. *Computer Networks*, 54(15):2711–2727, 2010.

[21] mod_antiloris. https://sourceforge.net/projects/mod-antiloris/.

[22] mod_limitipconn. http://dominia.org/djao/limitipconn.html.

[23] mod_reqtimeout. https://httpd.apache.org/docs/trunk/mod/mod_reqtimeout.html.

[24] M. Mongelli, M. Aiello, E. Cambiaso, and G. Papaleo. Detection of DoS Attacks through Fourier Transform and Mutual Information. In *2015 IEEE International Conference on Communications (ICC)*, pages 7204–7209. IEEE, 2015.

[25] D. Moustis and P. Kotzanikolaou. Evaluating Security Controls Against Http-based DDoS Attacks. In *2013 Fourth International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–6, 2013.

[26] Nginx Technical Specifications. https://www.nginx.com/products/technical-specs/.

[27] Rudy. https://packetstormsecurity.com/files/97738/r-u-dead-yet-denial-of-service-tool-2.2.html.

[28] A. Shevtekar and N. Ansari. A Proactive Test based Differentiation Technique to Mitigate Low Rate DoS Attacks. In *Proceedings of 16th International Conference on Computer Communications and Networks, 2007. ICCCN 2007.*, pages 639–644. IEEE, 2007.

[29] Slowloris. https://github.com/llaera/slowloris.pl.

[30] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An Overview of IP Flow-based Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010.

[31] N. Web Survey 2015. http://news.netcraft.com/archives/2015/06/25 /june-2015-web-server-survey.html.

[32] Y. Xiang, K. Li, and W. Zhou. Low-Rate DDoS Attacks Detection and Traceback by Using New Information Metrics. *IEEE Transactions on Information Forensics and Security*, 6(2):426–437, 2011.

[33] Y. Xie and S. Z. Yu. Monitoring the Application-Layer DDoS Attacks for Popular Websites. *IEEE/ACM Transactions on Networking*, 17(1):15–25, 2009.

[34] T. Yatagai, T. Isohara, and I. Sasase. Detection of HTTP-GET Flood Attack Based on Analysis of Page Access Behavior. In *2007 Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*, pages 232–235, 2007.