VOL 1 (2017) NO 4

e-ISSN : 2549-9904
ISSN : 2549-9610

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

A Review of Defense Against Slow HTTP Attack

Suroto [#]

[#] *Department of Information System, Faculty of Engineering, Batam University, Batam, Indonesia*
E-mail: zhuroto@yahoo.co.uk

***Abstract*— Every web server poses a risk to network security threats. One of them is a threat of Slow HTTP Attack. Slow HTTP Attack exploits the working methods of the HTTP protocol, where it requires that every request from the client be fully accepted by the server before it is processed. If the HTTP request is incomplete, or if the transfer rate is very low, the server remains busy waiting for the rest of the data. If the server is storing too many busy resources, there is a denial of service. Internet users can exploit such vulnerabilities, send incomplete data packets deliberately and requests repeatedly. When a web server is in a public network or the Internet, then protecting the computer and network security is an important issue. After identifying and analyzing how the Slow HTTP attack works, as well as its attack detection, this paper describes a portfolio of the work system, how to detect and how to defense against the Slow HTTP attack.**

***Keywords*— Slow HTTP Attack, Web Server Exploit, Denial of Service, DoS**

I. Introduction

Along with the rapid development of network technology, also increased threats to network or computer security. So that emerged various types of threats or attacks against computers and networks. One of attack types is a denial of service attack (DoS Attack). A DoS attack is a security intrusion action by attackers aimed to prevents legitimate users from accessing targeted host or other network resources.

Denial of Service Attack is generally made by flooding the server or host so that the victim's host runs out of resources (memory, CPU, traffic). This condition makes it unable to serve other users. Flooding is difficult to overcome, not enough just by rebooting, like other attacks. There are several variants of DoS attack. No less than 35 variants of this attack.[99] Each variant has different characteristics in terms of its attack. But they have the same effect, giving rise to a denial of service.

Slow HTTP attack is one of them. Slow HTTP Attack exploits the working methods of the HTTP protocol, where it requires that every request from the client be fully accepted by the server before it is processed. If the HTTP request is incomplete, or if the transfer rate is very low, the server remains busy waiting for the rest of the data. If the server is storing too many busy resources, then this creates a denial of service.

This enables an attacker to restrict access to a specific server with the very low utilization of bandwidth. This breed of DoS attack is starkly different from other DoS attacks such as SYN flood attacks which misuse the TCP SYN (synchronization) segment during a TCP three-way-handshake[4].

Therefore knowledge of what is a Slow HTTP attack, the types of attacks, how it works and the existing defense methods, becomes an important thing to have by anyone who works in the area of network security.

This paper aimed to review existing literature on defense against Slow HTTP attack. The structure of this paper has been organized as follows. Section I discusses a few papers background. Section II is dedicated to discuss comprehensive, relevant literature survey of existing defense against slow HTTP attack and elaboration of the practical usability of the defense system. Section III shows the findings made from the theoretical study and a brief discussion of key points about the different type of defense method and Section IV concludes the paper with attention on the theoretical analysis made on the ways of defense.

II. Material and Method

A basic understanding of HTTP, the Denial of Service (DoS), Slow HTTP, are necessary.

A. HTTP

One of the most popular application protocol used on the Internet is HTTP. HTTP stands for "Hypertext Transfer Protocol." HTTP is an application protocol that runs on top of the TCP/IP suite of protocols. The entire World Wide

Web uses this protocol. When we opened a web page, our browser probably has sent over 40 HTTP requests and received HTTP responses for each[10]. An HTTP headers are the core part of these HTTP requests and responses, and they carry information about the client browser, the requested page, the server and more[10].

As illustrated in figure 1, an HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message.

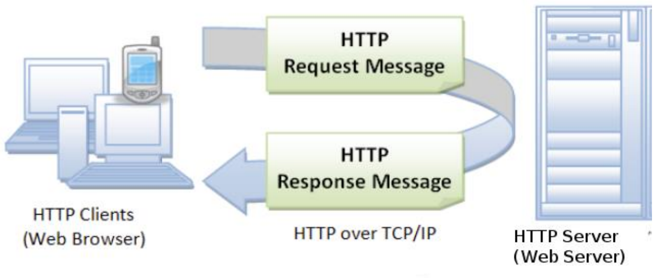


Fig. 1 HTTP Request message and response message

A program on the client side, called 'browser' will perform HTTP request to the server. The web browser is an HTTP client. Any Web server machine contains web page files (text, graphic images, sound, video, and other multimedia files) and also an HTTP daemon, a program that is designed to wait for HTTP requests and handle them. The client needs to type the correct Uniform Resource Locator (URL) address in the browser program or clicking on a hypertext link to get a web page or file, e.g., http://www.example.com/index.html. Then the browser converts the URL into a request message and sends it to the HTTP server. The HTTP daemon server receives and interprets the request message, and returns equested file or files associated with the request. This process is illustrated in figure 2 below:

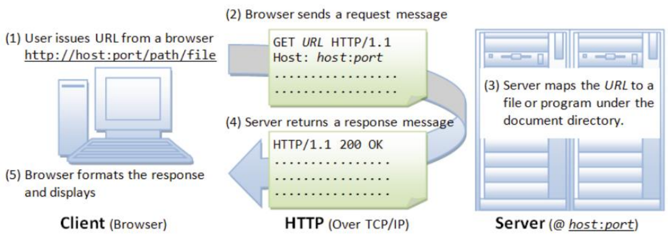

(2) Browser sends a request message
(1) User issues URL from a browser
http://host:port/path/file
GET URL HTTP/1.1
Host: host:port
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

Fig. 2 The process of communication between client and web server

As mentioned above, when client enters a URL in the address box of the browser, the browser translates the URL into an HTTP request message and sends it to the server.

For example, the browser translated the URL http://www.example.com/index.html into a request message, as shown figure 3 below:

```
GET /index.html HTTP/1.1
Host: www.example.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 11.0;)
(blank line)
```

Fig. 3 An HTTP GET request message from a translation of a URL

When this request message is received and interpreted by the server, the server will perform one of three actions:

- The server looks for the file under the server document directory and returns the requested file.
- The server runs the requested program and returns the program output to the client.
- The server returns an error message, because the request can not be fulfilled.

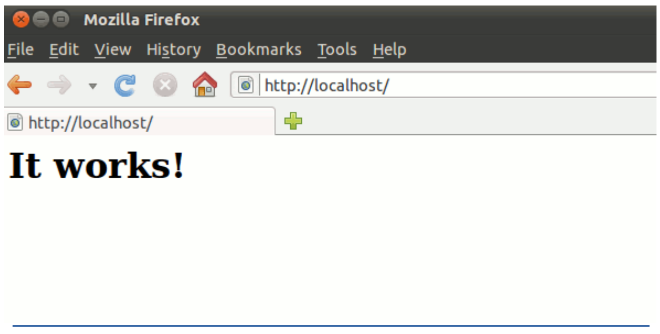
An example of the HTTP response message is as shown figure 4 below:

```
HTTP/1.1 200 OK
Date: Sun, 28 Aug 2017 08:56:53 GMT
Server: Apache/2.4.27 (Linux)
Last-Modified: Fri, 20 Jan 2017 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

Fig. 4 An HTTP response message

The browser program accepts, interprets and displays the contents of the response message in the browser window according to the Content Type. The example above, Content-Type is text/html. There are many content types, such as "text/text", "text/html", "audio/mpeg", "video/mpeg", "image/gif", "image/png", "image/jpeg", "application/pdf ", and others. Figure 5 illustrates a response message will appear in the browser window.



It works!

Fig. 5 An HTTP response message appears in browser

B. *Denial of Service*

Denial of service attacks are a security threat where an attacker sends a large number of fake requests to a host or server, so the target host deny access from authorized users. Service from host becomes unavailable. Therefore, the attack compromises the system's availability. Denial of service attacks (DoS) which aims at legitimate users, clients,customers from successfully accessing the internet has posing a serious challenge to the network security [26]. If a denial of service attack is launched from multiple computers, it is often called a Distributed Denial of Service

(DDoS) attack. The most commonly used DoS attack now is the DDoS attack where a large number of computers send thousands of requests to the system being attacked [28]. DDoS attacks occur when multiple hosts are infected with malware that allows hosts to be taken over by attackers; then the attacker program instructs them to access the target website. Usually, the host that is the target of the DoS attack is the web base system or web server. Generally, a web server programs, such as Apache, IIS have the ability to handle or receive many connections from users. Attackers benefit from the fact.

Due to the wide variety of attacks, it is helpful to classify them in order to clarify the process of defending against DoS. Attacks can take advantage of bugs or software weaknesses of routers and other network devices. In addition, vulnerabilities in the way operating systems implement protocols as well as in applications running on the victim machines may be exploited [5]. The classification of DoS attacks is shown in Figure 6.

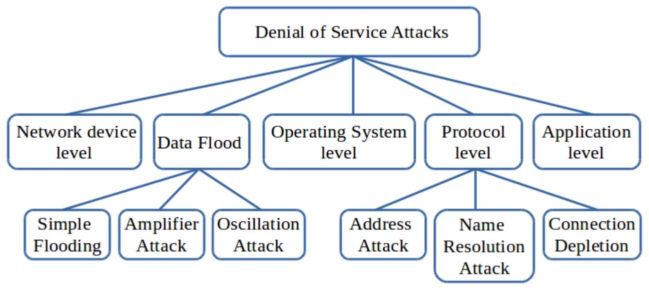


Fig. 6 The classification of Denial of Service Attacks

C. *Slow HTTP Attack*

Slow HTTP Denial of Service (DoS) is an application layer DoS attack in which a large number of incomplete HTTP requests are sent [8]. It is a layer 7 DoS. Application DoS attacks is a new class of DoS attacks which exploits the flaws in either application design or its implementation [6]. These attacks are harder to trace than Classical Dos attacks because

- These attacks do not consume a huge amount of bandwidth.
- These target on creating bottlenecks and resource limitation within the application by focusing on the weakest link in the application.
- These attacks normally use https as their transport to hide their true origin.

Slow HTTP attacks are primarily of three types [3] as follows :

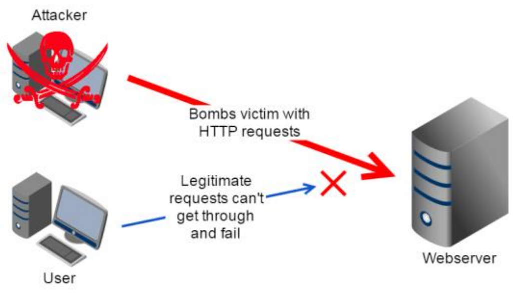
1) *Slow Headers (a.k.a Slowloris):* In the Slow Header attack, an attacker launches the action with the help of a tool called Slowloris or similar. This tool opens connections, then sending HTTP headers, augmenting but never completing the request. Thousands of HTTP POST connections are created and sends HTTP Headers very slowly to force the target web server to keep the connections open. This connection will remain alive, not disconnected from the target server. Slowloris will take all the resources from the target web server just for it, thus blocking requests from legitimate clients.

2) *Slow Body (a.k.a R-U-Dead-Yet):* Slow Body attack works just like Slow Header. An attacker with the help of a tool called R-U-Dead-Yet or similar sends a POST Body that will not end. The attack stage begins by making an initial TCP connection to the target web server. It then sends the HTTP POST header first as the normal connection does. A header contains the size information of the body of the data packet to be sent next. Attacker sends the message body with a very low speed. But the connection remains alive, making the victim's web server wait long enough. New and similar connections are created in large quantities, using all server resources and making legitimate connections impossible.

3) *Slow Read:* In a Slow Read Attack, attackers send valid TCP-SYN packets to opening a connection with the target's server. Then valid sessions established between them. Next, it begins to request a document from the target's server. Once the download begins the attacker's host begins to slow down the reading of received packets. This condition will continue and take all resources of the target's server. Slow Read Attacks are always non-spoofed in order to hold sessions open for long periods of time.

D. *How Slow HTTP Attack Work*

As described in the above section, attacks are performed with the help of a program script, which is able to transmit a partial request of the packet data , keeping multiple connections to the victim's web server open. Periodically, it sends the next HTTP header, but deliberately never complete. It triggers the victim's webserver to provide all of its resources for the attacker, which ultimately deny connections from legitimate users. An attacker doesn't need a huge bandwidth to take down the victim's webserver, but only create a large number of connections [2]. Figure 7 illustrates the attacks.



Attacker
Bombs victim with HTTP requests
Legitimate requests can't get through and fail
User
Webserver

Fig. 7 Illustration of Slow HTTP Attack

A 'request' message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use[11]. HTTP protocol defines a set of request methods. The methods are HEAD, GET, POST, PUT, DELETE, TRACE, CONNECT and OPTION.

We can perform an analysis of an HTTP GET request from the client to the web server. This analysis will assist in further explanation of HTTP GET requests. A tool, such as Firebug, Live HTTP Headers is needed to aid the analysis[10]. An extract of complete the HTTP GET request as shown figure 8 below:

```
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]
```

Fig. 8 Complete header of HTTP request

The example above is a normal GET Header. Each line of messages ends in a character CRLF. CRLF stands for CR (Carriage Return) and LF (Line Feed). The CRLF is a non-printable character. The request message ends with a blank line. There are two CRLF characters in the bottom row. They together are used to denote a blank line. The [CRLF] at the end of the request message attracts the attention of the attacker.

In the Slow HTTP attack, a blank line will never exist. The attacker deliberately did not send a CRLF character, at the end of the request. A request message as following this will result in Slow HTTP attack, because of the presence of single CRLF tag at the end denoting the header is incomplete, and server needs to wait for the complete header. The incomplete header samples are shown in figure 9 below:

```
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]
```

Fig. 9 Incomplete header of HTTP request by Slow HTTP Attack

E. *Identify Slow HTTP Attacks*

The effect of Slow HTTP attacks is that all clients can not connect to the web server. The site won't load and our clients will never get to see the content of web page. If our web server are under attack, we will see many connections on port 80 from source IP. The netstat command can show list connections as follows :

```
$ netstat -nalt | grep :80
tcp  0  0   0.0.0.0:80          0.0.0.0:*         LISTEN
tcp  0  0  24.34.0.1:4840    22.50.19.04:80    ESTABLISHED
tcp  0  0  24.34.0.1:4841    22.50.19.04:80    ESTABLISHED
tcp  0  0  24.34.0.1:4839    22.50.19.04:80    CLOSE_WAIT
tcp  0  0  24.34.0.1:4838    22.50.19.04:80    CLOSE_WAIT
tcp  0  0  24.34.0.1:4808    22.50.19.04:80    ESTABLISHED
tcp  0  0  24.34.0.1:4898    22.50.19.04:80    CLOSE_WAIT
tcp  0  0  24.34.0.1:4890    22.50.19.04:80    ESTABLISHED
```

On the server log will show the number of connections. Example, on Apache server, status looks like the following :

```
$ apachectl status
...
   CPU Usage: u3.1 s.2 cu0 cs0 - 1.0% CPU load
  .913 requests/sec - 31.1 kB/second - 17.5 kB/request
  611 requests currently being processed, 2 idle workers
….
```

The information above shows very low CPU usage, a lot of Apache processes, very few new requests. Attacker works by making many requests and more until it reaches Apache's MaxClients limit. If we look Apache's log, it will like this:

```
$ cat /var/log/httpd/error.log
   [mpm_prefork:error] [pid 1842] AH00161: server reached
MaxRequestWorkers setting, consider raising the
MaxRequestWorkers setting
```

For identify the IP address of attacker's machine, we can use netstat, a tool for view the most active IPs on server. Examples of using netstat as follows:

```
  $ netstat -ntu -4 -6 |  awk '/^tcp/{ print $5 }' | sed -r 's/:[0-9]+$//'
| sort | uniq -c | sort -n
```

This command will filter all the IPs that are connected to the server, order them and then count each unique occurrence. The output like this :

```
78   201.123.9.9
64   129.10.20.11
185  192.143.24.24
 ……
```

then after we know the attacker's IPs, blocking can be done on the IP address. On linux IPTABLE program is available. The following commands can be used:

```
$ iptables -A INPUT -i eth0 -s 201.12.9.9 -j DROP
```

the meaning of the command line above, if there is an incoming connection through interface eth0, from source's IP 201.12.9.9., then disconnect (DROP).

Early detection can be implemented by testing the web weakness or test web vulnerability. Many application for web vulnerability scanner is available on the world, such as Acunetix, OWASP and etc.

F. *Related Work*

Risk of computer security threats by Slow HTTP attacks has triggered the network security experts to develop the defense techniques. Many different methods & techniques of defense are available for webserver. These methods vary depending on the protected object and the set of rules in operation. Researchers from academics have also done a lot of research to detect & deal with Slow HTTP attacks. A large amount of literature is available in these attacks. Here, we summarize models or methods that focus on dealing Slow HTTP attacks.

Reference [8] proposed detection system is an anomaly detection system which measures the Hellinger Distance between two probability distributions generated in training and testing phases. Testing of proposed detection system by collecting simulated HTTP traffic on LAN and the Internet. Results show that proposed system detects Slow Header and Slow Message Body attacks with high accuracy.

Reference [12] analyze Slow Read DoS attack. Results that the efficient attack can be realized when the bandwidth is over 500 Kbps. Also, the researcher found the secure setting of web server against Slow Read DoS attack.

Reference [13] study slow DoS attacks, analysing in detail the current threats and presenting a proper definition and categorisation for such attacks. The research aimed to provide a useful framework for the study of this field, and for the proposal of innovative intrusion detection methodologies.

Reference [14] analyzed the effectiveness of Slow Read DoS Attack by virtual network environment. The research concluded that attacking by a single attacker is not so efficient. A secure module for a web server, ModSecurity can limit the length of attack success status.

Reference [15] propose and evaluate a defense method against Distributed Slow HTTP DoS attack by disconnecting the attack connections selectively by focusing on the number of connections for each IP address and the duration time.

Reference [16] designed an attack tool, named SlowDroid. A tool runs on an Android mobile device. The research compares attacks with similar tools that already exist. The results that the tool designed is a serious threat.

Reference [17] analysing web-server behaviour when various types of Slow HTTP-attack occurs using mathematical models. Analyse aimed at building the model for the systems for the types of Slow HTTP attacks detection.

III. Results and Discussion

A. Defense against Slow HTTP Attacks

Defense against HTTP attacks is made with a particular configuration, so that attacks can be prevented or reduced. Prevention can be done in general and specific configuration. The general configuration can be applied to machines running any web server application (Apache, Nginx, and others). While the specific configuration is applied to a particular web server. For example, the configuration for preventing Slow HTTP attacks on Apache will be different from Nginx.

The general configuration aimed to prevent Denial of Service (DoS) attacks against a network service. In Linux, we use the xinetd daemon. The xinetd daemon can add a basic level of protection from Denial of Service (DoS) attacks. We must configure file /etc/xinetd.conf or create a new file in the /etc/xinetd.d directory and add some command line. The following is a sample config file for service called http located at /etc/xinetd.d/ http.

```
$ nano /etc/xinetd.conf
```

or

```
$ nano /etc/xinetd.d/http
```

Then append following text at the end of the file :

```
service http
{
   protocol = tcp
   server = /usr/sbin/apache2
   cps = 15 20
   instances = 5
   per_source = 2
   max_load = 3.0
}
```

Where,

service: Specifies the service name. In this case, the service to be protected is http.

protocol: Sets the protocol type to TCP.

server : the running server program.

cps = 15 20 : Limit to 15 connections per second. If the limit is exceeded, sleep for 20 seconds.

Instances = 4 : Limit to 4 concurrent instances of myservice.

per_source = 2 : Limit to 2 simultaneous sessions per source IP address. The default is UNLIMITED .

After setting the configuration, the xinetd service needs to be restarted. To restart xinetd service, type the command:

```
$ /etc/init.d/xinetd restart
```

B. Defense on Nginx Web Server

Nginx has provided some configuration parameters to prevent and mitigate Slow HTTP attacks. This configuration is stored in the nginx.conf file. Nginx has features to prevent and mitigate such attacks, by :

- controlling buffer overflow attacks.
- controlling timeouts.
- controlling simultaneous connections.

For controlling buffer overflow attacks, edit the nginx.conf file, usually in /usr/local/nginx/conf/ directory.

```
# cd /usr/local/nginx/conf
# nano nginx.conf
```

and append following text to set the buffer size limitations for all clients as follows:

```
client_body_buffer_size  1K
client_header_buffer_size 1k
client_max_body_size 1k
large_client_header_buffers 2 1k
```

Where,

client_body_buffer_size 1k : sets the client request body buffer size.

client_header_buffer_size 1k : sets the header buffer size for the request header from client.

client_max_body_size 1k : sets the maximum accepted body size of the client request.

large_client_header_buffers 2 1k sets the maximum number and size of buffers for large headers to read from client request. 2 x 1 k will accept 2 kiloByte data URI.

We also need to control timeouts to improve server performance. Lower the timed out wait time for each http connection. Append the following text into nginx.conf file :

```
client_body_timeout   12
client_header_timeout 12;
keepalive_timeout     15;
send_timeout          10;
```

where,

client_body_timeout : to close the connections with slow body

client_header_timeout : to close the connections with slow headers

send_timeout : If the client does not receive anything within this time, the connection is closed.

For controlling simultaneous connections, Nginx provide HttpLimitZone module. Edit nginx.conf and append following command line :

```
limit_zone slimits $binary_remote_addr 5m;
limit_conn slimits 5;
```

Where,

limit_zone slimits $binary_remote_addr 5m; limit the number of simultaneous connections for the assigned session from single IP address.

limit_conn slimits 5 : limits remote clients to no more than 5 concurrently “open” connections per single ip address.

All configuration above can provide protection, prevention and mitigate Nginx web server from Slow HTTP attacks.

C. *Defense on Apache Web Server*

Many methods or techniques can be applied to prevent and mitigate Slow HTTP attacks on Apache server. We can take advantage of the built-in features of the operating system and Apache itself. In Apache Web Server version 2.2.15 or above, some modules are available, such as mod_reqtimeout, mod_qos, mod_security, and mod_antiloris.

1) *Using mod_reqtimeout:* The mod_reqtimeout module allows we to set a time limit for receiving HTTP request headers and body from clients. Therefore, if header or body data is not received by the Apache server within a specified time, the 408 REQUEST TIME OUT error message is sent by the server[4]. An example of an Apache configuration with enabled mod_reqtimeout module as follow:

```
<IfModule mod_reqtimeout.c>
  RequestReadTimeout header=15-20,MinRate=512 body=15,MinRate=512
</IfModule>
```

Put the above command line inside /etc/apache2/httpd.conf and restart Apache. The above Apache configuration will allows a client send for the first byte of the request line+headers in 15 seconds and maximum 20 seconds for the headers to complete. Allows a client sends header data at a rate of 512 bytes per second. Then Apache server will allow the client to send body data for up to 15 seconds and up to 20 seconds for the body of the request to complete. Result: stopped the attacker after 15 seconds , but allowed to attacker to retry after x time.

2) *Using mod_qos:* The next module is mod_qos. The mod_qos is a quality of service (QoS) module for the Apache HTTP Server. It is used to reject requests to unimportant resources while granting access to more important applications. It provide control mechanisms base on levels of priority to different HTTP requests. An example of configuration mod_qos to prevent Slow HTTP attacks as follows :

```
<IfModule mod_qos.c>
QS_ClientEntries 500
QS_SrvMaxConnPerIP 10
MaxClients 200
QS_SrvMaxConnClose 70%
QS_SrvMinDataRate 150 1200
</IfModule>
```

Put the above command line inside httpd.conf and restart apache. The above configuration allows the server to track up to 500 connections and allow the server to receive up to 200 connections only. In addition, each IP address is allowed to make simultaneous connections up to a maximum of 10 connections. HTTP KeepAlive will switch off when 70% of connections are used. The configuration requires a minimum of 150 bps per connection, and 1200 bps when MaxClients is reached. Result : stopped the atack after 1 second and stopt the attacker from using the same IP.

3) *Using mod_security:* The mod_security module is a free Web Application Firewall (WAF) that works with Apache, Nginx, and IIS. This module is used to protect a website from various attacks such as XSS, LFI, SQL-injection, Password attack Trojans, session hijacking and much more. This module can be applied to carry out specific functions. The configuration is done by editing the file /etc/modsecurity/modsecurity.conf. This file is a copy of the original file /etc/modsecurity/modsecurity.conf-recommended. When installing mod_qos, the config file is created automatically. Step configuration to mitigate a Slow HTTP attack as follows: Edit the modsecurity.conf file.

```
# nano /etc/modsecurity/modsecurity.conf
```

Then find this line:

```
SecRuleEngine DetectionOnly
```

and change it to:

```
SecRuleEngine on
```

The mod_security module requires rules to work. It have security rules, called Core Rule Set (CRS), located at /usr/share/modsecurity-crs directory. We will create a rule chain which blocks the request of attacker. Custom rules can be created in a separate new file and placed in modsecurity directories.

```
# nano modsecurity_SlowHTTP_rules.conf
```

and put the following rule line inside the *.conf file:

```
SecAction
phase:1,id:122,nolog,pass,initcol:ip=%{REMOTE_ADDR}
SecRule RESPONSE_STATUS "@streq 408"
"phase:5,t:none,nolog,pass, setvar:ip.slow_dos_counter=+1,
expirevar:ip.slow_dos_counter=60, id:'123'"
SecRule IP:SLOW_DOS_COUNTER "@gt 6"
"phase:1,t:none,log,drop,msg:'Client connection dropped due to
suspected as  an slow http attack', id:'124'"
```

The above rule record how many times the IP address has triggered a 408 error code. If this event has happened more than 6 times in 60 seconds, the next request for that IP address will be dropped by mod_security. Clients can still connect again after 6 minutes.

4) *mod_antiloris apache module:* Another solution, use an apache module called as mod_antiloris. This module will protect Apache 2.x from the slowloris attack. The module limits the number of simultaneous connections per IP address that are in READ state. The following command line to enable mod_antiloris :

```
# apxs -a -i -c mod_antiloris.c
```

then restart Apache:

```
# service httpd restart
```

finally, check whether mod_antiloris loaded or not:

```
# httpd -M | grep antiloris
```

output like this:

```
antiloris_module (shared)
```

Finally, mod_antiloris have protect Apache web server from Slowloris attacks.

D. Defense on IIS Web Server

Just like Apache and Nginx, IIS also has some features and modules that can be adjusted to reduce this attack. The configuration settings are stored in the web.config file. Microsoft IIS has provided Internet Information Services (IIS) Manager, a GUI for easy IIS server management and settings.

1) WebLimits: The <webLimits> element can help preventing Slow HTTP attacks on IIS Server. WebLimits has some attributes, such as: connectionTimeout, dynamicIdleThreshold, headerWaitTimeout and minBytesPerSecond. The connectionTimeout attribute is used to sets the waiting time for IIS, before disconnecting a client considered inactive. The dynamicIdleThreshold attribute is used to sets the percentage of committed physical RAM. The headerWaitTimeout attribute is used to sets the time that IIS waits for all HTTP headers request before disconnecting a client connection. The minBytesPerSecond attribute is used to specifies the minimum throughput rate, when server sends a response to the client. If the throughput rate is lower than the minimum byte setting, the connection is terminated. Below are an configuration which using webLimit:

```
<configuration>
  <system.applicationHost>
    <webLimits connectionTimeout="00:00:45"
      headerWaitTimeout="00:00:35"
      dynamicIdleThreshold="150"
      minBytesPerSecond="512"
    />
  </system.applicationHost>
</configuration>
```

The above configuration specifies that the connection time-out to 45 second, the header wait time out to 35 seconds, the percentage of committed RAM to 150. Finally, IIS allows the minimum throughput rate to 512 bytes per second.

2) Request Limits: The <requestLimits> element sets limits on HTTP requests that are processed by the IIS. These limits such as: the maximum length of the URL, the maximum length of the query string, the maximum length of content in request and size limits for HTML headers. These limits are specified in attributes of <requestLimits>, such as maxAllowedContentLength, maxQueryString, and maxUrlLength. The following is an code sample of the <requestLimits> element.

```
<configuration>
  <system.web>
    <httpRuntime maxUrlLength="2048"
            maxQueryStringLength="1024" />
  </system.web>
</configuration>
```

The above code will configure IIS to deny access for HTTP requests where the length of the URL is greater than 2048 bytes (2KB) and the length of the query string   is greater than 1024 bytes. While, the below codes will configure IIS to drop for HTTP requests where the length of the "Content-type" header is greater than 100 bytes.

```
    <configuration>
      <system.webServer>
  <security>
    <requestFiltering>
      <requestLimits>
        <headerLimits>
        <add header="Content-type" sizeLimit="100" />
        </headerLimits>
      </requestLimits>
    </requestFiltering>
    </security>
  </system.webServer>
</configuration>
```

The combination of these two codes can reduce Slow HTTP attacks to IIS Server.

IV. Conclusions

The Slow HTTP attacks can be just as cruel as other DDoS attacks, if not handled properly. Each web server has its own properties and requires special handling. There are many configuration options for each web server. We do not need to use all of these options to apply to the server. Just one or two choices. But it must be considered in the selection of methods / techniques, so that the configuration is not overlapping or opposite, so even ineffective.

Finally, We know that Slow HTTP attacks can be prevented and even eliminated, if the web server is installed the right security system. We can also see that this paper has thoroughly discussed how to handle this attacks on some famous web servers.

Acknowledgment

The author would like to thank Faculty of Technology, Batam University for permits to use Laboratory of Computer and Internet connection when write this paper.

References

[1] A. Nicolic. (2013) The nmap website. [Online]. Available: https://nmap.org/nsedoc/scripts/http-slowloris-check.html

[2] T. Mansoor. (2012). The admin-ahead website. [Online]. Available: https://admin-ahead.com/blog/analyzing-the-anatomy-of-a-dos-attack-using-slowloris/

[3] S. Kumar. (2012). The Geeks website. [Online]. Available: http://www.geeksforgeeks.org/slow-http-can-knock-server/

[4] I. Muscat. (2013). The Acuanetix website. [Online]. Available: http://www.acunetix.com/blog/articles/slow-http-dos-attacks-mitigate-apache-http-server/

[5] S. Ramanauskaite, A.Cenys "Taxonomy of DoS attacks and their countermeasures " *Central European Journal of Computer Science*. Vol 1, Issue 3, pp. 355-366, Sept. 2011

[6] D Sai Krishna et al,"Application Denial of Service Attacks Detection using Group Testing Based Approach". *International Journal of Computer Science & Communication Networks*,Vol 2(2), pp. 167-171, Feb. 2012

[7] I. Sommerville, *Software Engineering*, 10nd ed. Essex – England: Pearson, 2015

[8] N. Tripathi, et al. "How Secure are Web Servers? An Empirical Study of Slow HTTP DoS Attacks and Detection", in *Reliability and Security (ARES), 2016*, pp. 454–463

[9] I. Muscat. (2017) The Acuanetix homepage. [Online]. Available: https://www.acunetix.com/blog/docs/http-sniffer/

[10] B. Gussel. (2009) The tutsplus homeepage. [Online]. Available: https://code.tutsplus.com/tutorials/http-headers-for-dummies--net-8039

[11] (2017) The W3 website. [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html

[12] Tayama S., Tanaka H, "Analysis of Slow Read DoS Attack and Communication Environment", in *International Conference on Mobile and Wireless Technology, ICMWT*, 2017, p. 350-359.

[13] E. Cambiaso, G. Papaleo, G. Chiola, et al, "Slow DoS attacks: definition and categorisation", *International Journal of Trust Management in Computing and Communications (IJTMCC)*, Vol. 1, pp. 300-319, Sept 2013.

[14] J. Park, K. Iwai, H. Tanaka and T. Kurokawa, "Analysis of Slow Read DoS Attack and Countermeasures on Web servers", *International Journal of Cyber-Security and Digital Forensics (IJCSDF)* Vol. 4(2): pp. 339-353, Sept 2015.

[15] T. Hirakawa, K. Ogura, B. Bahadur and T. Takata, "A Defense Method against Distributed Slow HTTP DoS Attack", in NBiS, 2016, p. 152-158.

[16] E. Cambiaso, G. Papaleo, G. Chiola and M. Aiello, "Mobile executions of Slow DoS Attacks", *Logic Journal of the IGPL*, Vol. 24, Issue 1, pp. 54–67, Feb 2016.

[17] I. Duravkin, A. Loktionova and A. Carlsson, "Method of slow-attack detection", in *Problems of Infocommunications Science and Technology*, 2014, p. 102-106.