A Practical Approach and Mitigation Techniques on Application Layer DDoS Attack in Web Server

Article *in* International Journal of Computer Applications · December 2015

DOI: 10.5120/ijca2015907209

3 authors, including:

Muhammad Yeasir Arafat
Chosun University
34 PUBLICATIONS **1,003** CITATIONS

SEE PROFILE

A Practical Approach and Mitigation Techniques on Application Layer DDoS Attack in Web Server

Muhammad Yeasir Arafat
Department of Electrical and Electronic Engineering
School of Engineering and Computer ScienceIndependent University, Bangladesh

Muhammad Morshed Alam
Department of Electrical and Electronic EngineeringIslamicUniversity ofTechnology (IUT), Dhaka, Bangladesh

Mohammad Fakrul Alam
Bdhub Limited,
Dhaka, Bangladesh

ABSTRACT
Denial of Service (DoS) or Distributed Denial of Service (DDoS) is a powerful attack which prevents the system from providing services to its legitimate users. Several approaches exist to filter network-level attacks, but application-level attacks are harder to detect at the host base firewall. Filtering in application level can be computationally expensive and difficult to scale, while DDoS attacks still creating bogus positives that block legitimate users. In this paper, the authors show application layer DoS attack for HTTP web server using some open source DoS attack tools and also suggest some realistic mechanisms that can protect a web server from application-level DoS attacks especially while attacks targeting the resources including CPU, sockets, memory of the victim server. The authors propose a new DDoS defense mechanism that protects http web servers from application-level DDoS attacks based on the reverse proxy. The attack flow detection mechanism detects attack flows based on the symptom or stress at the server, since it is getting more difficult to identify bad flows only based on the incoming traffic patterns. A popular software known as Wireshark which is a network protocol analyzer is used to capture the packets during a DoS attack from the victim server Ethernet interface to detect the attacking host IP address and analysis the types of attack. We evaluate the performance of the proposed scheme via experiment.

Keywords
HTTP, TCP, Slowloris, OWASP, OSI layer attack; Nginx, fail2ban, IPtables.

1. INTRODUCTION
DoS attack is a malicious attempt to disrupt the service provided by networks or servers. The power of a DoS attack is amplified by incorporating over thousands of zombie machines through bonnets [1] and mounting a DDoS attack. Leveraging botnets and high-speed network technologies, modern DoS attacks exceed the scale of 300 Gbps becoming a major threat on the Internet [2]. Being one of the oldest type of attacks on the Internet, DoS attacks are known for their disruptiveness and ability to deplete the computing resources and/or bandwidth of their victims in a matter of minutes. Although many defense mechanisms have been proposed to counter DDoS attacks [3], this remains a difficult issue, especially because the attack traffic tends to mimic normal traffic recently.

If a small number of machines are participating in a DoS attack to a selected server, the IP addresses of those attack machines might be detected using the approaches[4] [5] of without managing per-flow states. However, if the number of machines participating in a DoS attack increase, each attack node needs not send attack traffic at a high rate, since the aggregate rate of attack traffic from many BOT nodes can be sufficiently high to cause critical damage to the target node. This kind of low rate DoS attacks may not be easily detected by conventional metrics of per-flow traffic rate or SYN packet rates, since such low rate attack traffic is not much different from the traffic of normal users in terms of those metrics. Thus, the decrease of the attack traffic rate due to the large population of attack machines recruited through a botnet is becoming a challenge for DDoS defense.

There is another factor that makes it more difficult to discriminate attack traffic of bots from the traffic of normal users. If DoS attack is launched at the application layer, then the attack can be effective with a small number of packets. For example, some specially crafted http request packets might induce an extensive database search, inject, or modify the data in the database disabling the target server ultimately. Slowloris, slow header, slow header slow post and ddosim, [6]-[8] are well-known tools that can launch network/transport layer DoS attacks as well as application layer DoS attacks such as http get flooding attack and CC attack.

These low rate application-level attacks may not be detected by conventional DoS detection mechanisms based on the SYN packet rate or traffic rate. In this paper, a new approach is investigated to detect this application-level DoS attacks, especially targeting http web servers. Recently emerging application-level DoS attacks may not be distinguished from normal user traffic. However, the intention of the attacking machines differs from that of normal users. Although normal users just want to get the information in which they are interested, malicious machines attempt to burden the target server as much as possible. Thus, we attempt to discriminate the attack flows from normal user flows based on the time interval during which each client makes the server busy. Since this step requires at least tens of seconds, this attack flow detection mechanism may be insufficient to protect a given web server in real time. Thus, we use an additional step of IP whitelist-based admission control to protect the given web server or server farm in real time.

The remainder of this paper is organized as follows. Authors first discuss related work in Section 2. In Section 3, authors showed different types DoS attack on the web server and also showed effect of the DoS attack on web server. In Section 4, authors propose a mitigation technique based on reverse proxy. In Section 4, the performance of the proposed DDoS defense mechanism is evaluated by experiment in linux based web server. Finally, conclusions are presented in Section 5.

2. RELATED WORK

Mirkovic et al. [9] has classified DoS attacks into two categories. The first type is the flooding attack, which targets overwhelming the resource of the victims, by sending a sufficiently large amount of traffic to the victims. The second type is the vulnerability attack, which takes advantage of a vulnerability in the victim and sends specially crafted messages to the victim to disable it. In this paper, we focus only on the first type of attack. Several types of low-rate DoS attacks have been reported recently. One example is shrew attack against TCP [10]. The attacker sends bursts of packets to create packet losses in a link and increments the retransmission timeout for certain TCP flows. The bursts are sent only around the expiration times of these flows to reduce the overall throughput. Another example is low-rate DoS attacks against application servers [11].

Regarding the defense against these low-rate DoS attacks, Sun et al. [12] reports that the ON/OFF traffic pattern of the Shrew attack can be detected using the autocorrelation of the traffic rate signal and dynamic time warping (DTW). Other researchers have tried to detect the attack, from analyzing the frequency information by spectral analysis [13] or by considering the correlation between the ON/OFF traffic rate signal and the round trip time of the affected flows [14]. However, since the attack traffic is generated by the attacker, there is a possibility that the attackers evade these traffic signatures-based detection mechanisms by changing the traffic pattern. Thus, we attempt to detect attack flows based on the symptoms appearing in the server, rather than based on the incoming traffic pattern.

Srivasta et al. [15] suggested a mechanism based on admission control and congestion control. In the admission control step, the client is required to solve a computational puzzle that is implemented through JavaScript. In the congestion control step, the server monitors the behavior of each flower to give a higher priority to well-behaving flows. When the behavior is monitored, the response time for each request packet is also considered. The packet response time is related to the metric of the busy period considered in this paper, but they are different, as described in the subsequent sections. In addition, the congestion control functions are performed in the server-side kernel or firewall. However, since these defense functions can be a burden to the server itself, we consider the defense mechanism that can protect a single server or server farm while running on a machine physically separated from the servers.

Ranjan et al. [16] tried to provide DDoS resilience to web servers by allocating suspicion measure to each session and scheduling the requests of each session based on the suspicion measure. Since the suspicion measure tries to capture the deviation of session behavior of the normal model, it is very important to set up a reliable normal model. However, the normal model construction is usually difficult, and normal model might be susceptible to pollution by the attackers. Ranjan's mechanism does not consider large scale attacks that involve a large number of attack sessions, but our proposed scheme can cope with such a large scale attack, because our mechanism registers malicious flows in a blacklist and drops the packets from the blacklisted IP addresses, instead of allowing them with a lower priority.

3. TYPES OF HTTP DOS ATTACK AND EFFECTS ON WEB SERVER

Application layer DDoS attacks employ legitimate HTTP requests to flood out victim's resources. Attackers attacking victim web servers by HTTP GET requests (HTTP flooding), HTTP post requests and pulling large image files. Sometimes attackers can run a large number of queries through victim's search engine or database query and bring the server down. To utilize the standard valid GET requests used to fetch information, as in typical URL data retrievals (images, information, etc.) Targeted server is barraged with basic GET requests. Generally Botnets are usually employed in HTTP GET flood attacks. HTTP GET flood attacks are hard to tell from valid traffic because they use standard URL requests. Slowloris used time-delayed HTTP headers to hold on to HTTP connections and exhaust web server threads or resources. This method employs HTTP POST requests used with forms whose entire set of headers is sent correctly by waiting for the complete message body to be sent, web servers can support users with slow or intermittent connections. Hence, any website which has formed, i.e. accepts HTTP POST requests, is susceptible to such attacks common uses of HTTP POST requests: login, uploading photo/video, sending webmail / attachments, submitting feedback and etc. Are You Dead Yet is used for launching HTTP Post DDoS Attack.

3.1 Slowloris HTTP DoS Attack

Slowloris is a GET-based DDoS tool on the impression of keeping the server busy with very few resources by allowing a single server to take down another web server with low bandwidth and side effects on unrelated services and ports. Slowloris holds connections open by sending slow, incomplete HTTP requests and also time-delayed HTTP refers headers to the victim web server and continues to send consequent headers at normal intervals to keep the sockets from closing. Slowloris is a highly-targeted attack, enabling one web server to take down another server, without affecting other services or ports on the target network. Slowloris does this by holding as many connections to the target web server open for as long as possible. It accomplishes this by creating connections to the target server, but sending only a partial request. Slowloris constantly sends more HTTP headers, but never completes a request. The targeted server keeps each of these false connections open. This eventually overflows the maximum concurrent connection pool and leads to denial of additional connections from legitimate clients.

The script is written by Rsnake. When the victim web server receives incomplete HTTP headers, it assumes that a client is on an unreliable and slow network, and the rest will arrive in disjointed packets. On the other hand the request that actually sent by slowloris tool usually never gets completed that cause unavailable of sockets to the legitimate users with no http acknowledgement. Instead of flooding the server with requests, it holds the links open for a very long time. While the slowloris attack is running, the log file will not be written to the victim server until the request is completed. When the attack stops or the session gets close there will be several 400 errors in the web server logs in Figure 1. Finally the victim web server connection pool will be entirely busy processing with httpd web application services to remain dedicated to the client attacking due to lots of incomplete http requests generated by slowloris tool and then it will start denying new connection attempts from genuine clients.

Slowloris script runs on Linux based operating systems (OS) like Centos or Ubuntu with a general format of the command given below:

[root@web~]# perl ./slowloris.pl –dns [www.testserver.com] –options

Here for the test and research purpose authors used slowloris script in centos based Linux kernel with host Internet Protocol (IP) address 123.200.0.38. The authors also choose a web server with IP address 123.200.0.36 that also runs at centos based Linux kernel which will be used as a victim web server that will face slow loris DOS attack from the host 123.200.0.38.

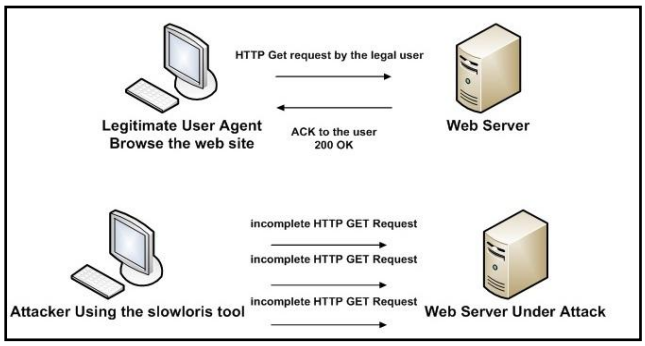

Fig. 1 Basic difference between normal http request and partial http request using slowloris tool

Here authors will discuss how the slowloris tools works and how rapidly it can down a web server to its legitimate users. Command to run the slowloris tool with domain or IP address 123.200.0.36 on the victim web server from the attack server with IP address 123.200.0.38 is given below:

[root@web~]#perl ./slowloris.pl -dns 123.200.0.36 -port 80 -timeout 1 -num 600000 –cache

After running this command slowloris script starts to send lots of packets known as slow, incomplete http get request with low bandwidth via building rapid sockets as we discussed earlier to the victim web server which is 123.200.0.36 where the slowloris script running OS with IP address 123.200.0.38 using the default http port 80 as like as below Figure 2.

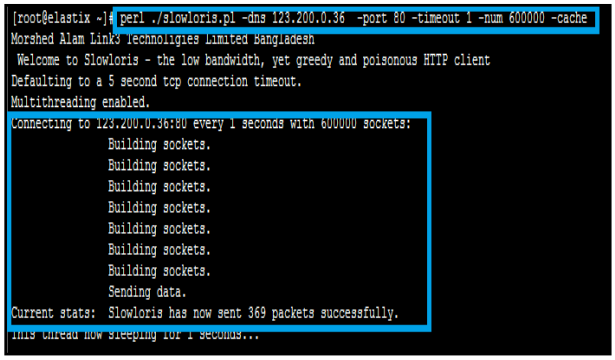

Fig.2 slowloris script running from the attacking server with IP address 123.200.0.38

When the script runs from the attacking server with IP address 123.200.0.38 for the detection on the attacking host authors run tcpdump command on the victim server with IP address 123.200.0.36 to see the packets that pass through its Ethernet interface 0 and save the file as slowlorisdetection.pcap in root directory using below command-

[root@web ~]#tcpdump -n -i eth0 -s 0 -w slowlorisdetection.pcap

Above pcap file gives the flow graph at wireshark which shows the victim server IP address 123.200.0.36 sends 400 http bad requests to the attacking server 123.200.0.38 as like as Figure 3 since the victim sever busy with handling incomplete slow http get request. Therefore, it causes denial of service (DoS) attack by using a very slow HTTP get requests to the victim site as slow as possible, the server is enforced to continue to wait for the complete headers to arrive. If enough connections are opened to the victim server in this format, it is quickly unable to handle legitimate requests. As slowloris increased the running process of service application httpd of Linux kernel it has a huge impact on CPU initialization percentage.

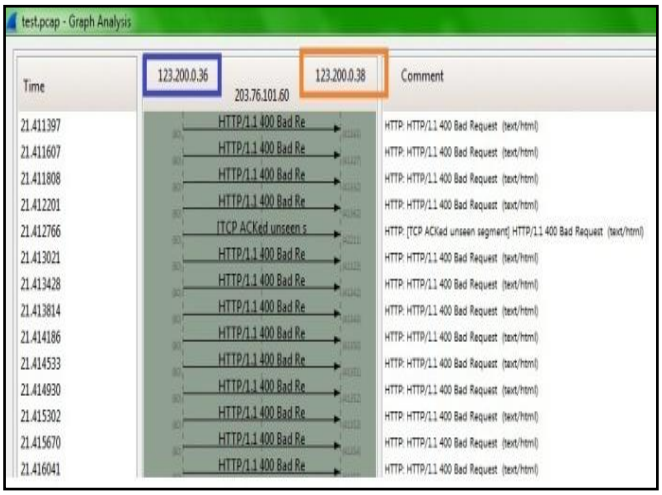

Fig. 3 400 HTTP bad requests to the attacking server 123.200.0.38 from the victim 123.200.0.36

When attacker runs slowloris tool as DoS attack it increases the CPU uses percentage of a victim machine as the service httpd process increased from the kernel. Service httpd running process at normal condition on the victim server with IP addresses 123.200.0.36 when a legitimate user login to the server.

3.2 OWASP HTTP Slow Header DoS Attack

For a web server HTTP headers contain significant information which is coming from the user, while the requests are processed, web server waits to capture complete request of http headers before processing for a message as an acknowledgment to the request sender. DoS attackers took the benefit of this behavior and produced lots of fake requests which keep sending incomplete headers and requests that never complete. Therefore server connections and memory resources are tied up with these incomplete requests. The Slow header attack works by exploiting the client idle timeout value on the victim web server. This timeout is configured at server side to drop a client connection if a client was found idle during the time period. The Slow header attack finds the estimated timeout value set on the victim server side and then chooses a value which is lower than the configured value. Then this attack generates an HTTP requests with partial header or incomplete header to the victim web server. It keeps sending one header based on the selected value such that client idle timeout will not be triggered on the victim web server and requests will not be completed. From above discussions, Slow HTTP POST attack is very similar to Slowloris. Here headers of HTTP POST requests are sent correctly, including the content-length. After the headers are sent and received, the POST message body is sent at a very low rate, thus keeping the connection open for an extended time. The server has to wait until all content arrives according to the declared content-length.

This tool provides graphical view and runs even on Windows operating systems. For the research purpose authors have used this tool in Windows-7 operating systems with IP address 202.4.96.197 and the victim server IP address is 202.4.96.198 which is centos based operating system with having a httpd web server. The graphical view of this tool is given Figure 4.

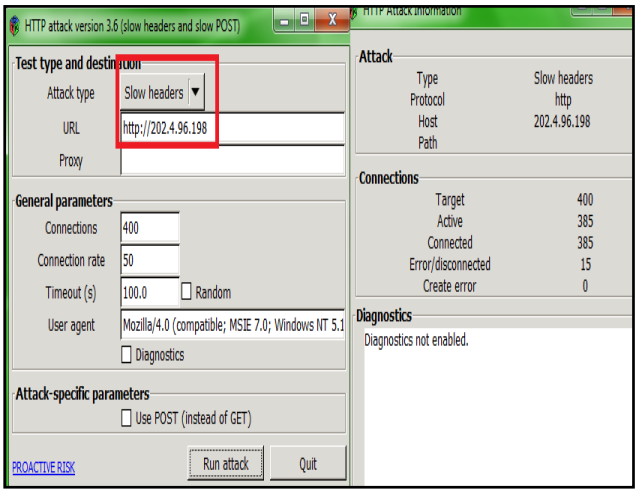

Fig.4 OWASP DoS attack tool for HTTP slow headers attack

When this attack runs, data traffics are captured from the victim web server Ethernet interface using wireshark which gives lots of partial http request packets mentioning a message that the continuation of headers or non-http traffic as given Figure 5.

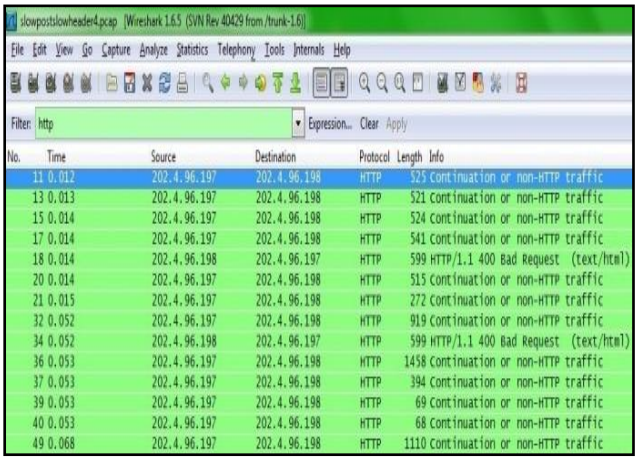

Fig.5 Received partial HTTP packets captured from victim Web server

Due to receive lots of incomplete http headers which never complete therefore victim web server httpd service application increases rapidly. The CPU utilization percentage and memory resources are tied up as discussed earlier. Below command is used for to see the httpd service running processes on victim web server during the attack.

3.3 HTTP Slow Post DoS Attack

The distinction between slow header and slow post HTTP DoS attack is that in case of slow header, the attack works with partial or incomplete HTTP header. On the other hand, slow post attack sends a full HTTP request header but sends partial data. Slow post DoS attack works by partial posting of data to the victim web server and keeping the socket connections always alive. HTTP slow-post DoS attack uses a common form of HTTP method used in most of applications, thus causing high memory and CPU resource utilization in the victim web server due to run lots of httpd service processes. DoS attacker using this tool sends a HTTP post request with large content-length header value. Due to this, the victim web server has to consider that the user is going to send so much of data as mentioned in the HTTP header. Therefore the victim web server keeps the connection open to receive content-length significance of data. But this DoS attack tool sends one byte of POST data at regular time interval configured by the attacker such that connection remains alive while this creates lots of service httpd processes at victim web server causing high memory and CPU utilization. Because of this client idle timeout will not be triggered and server keeps the connection alive till all the bytes of data specified in content-length header were received by the web server.

This tool provides graphical view and runs even on Windows operating systems. For the research purpose authors have used this tool in Windows-7 operating systems with IP address 202.4.96.197 and the victim we server IP address is 202.4.96.198 which is a Centos based operating system with having an httpd web server. The graphical view of this tool is given in Figure 6.

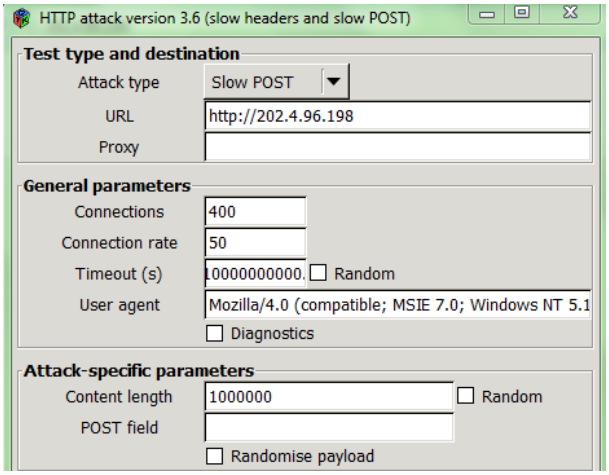

Fig.6 OWASP DoS attack tool for HTTP slow post attack

When this attack runs, data traffics are captured from the victim web server Ethernet interface using wireshark which gives lots of partial http request packets mentioning a message that the continuation of headers or non-http traffic as given Figure 7.

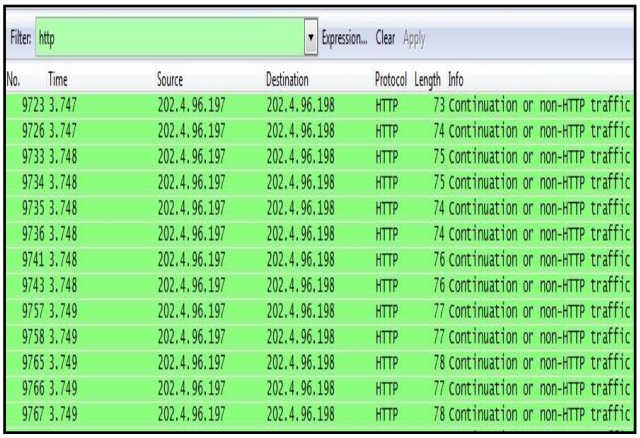

Fig.7 Received HTTP packets with partial data captured from the victim web server

This attack has significant effects of the memory and CPU utilization of victim web server as lots of service httpd web application processes to keep alive connections to get the complete data as declared large content-length http header value by the attacker in the attack-specific perimeters.

As the tool sent http headers with large content-length lots of traffic of http packets has been sent to the victim web server, however this tool sent one byte of POST data at regular time interval configured by the attacker. During this attack packet vs tick (X-axis shows Time, Y-axis shows packet) graph shows high traffic that captured from the victim server Ethernet interface as like as Figure 8.

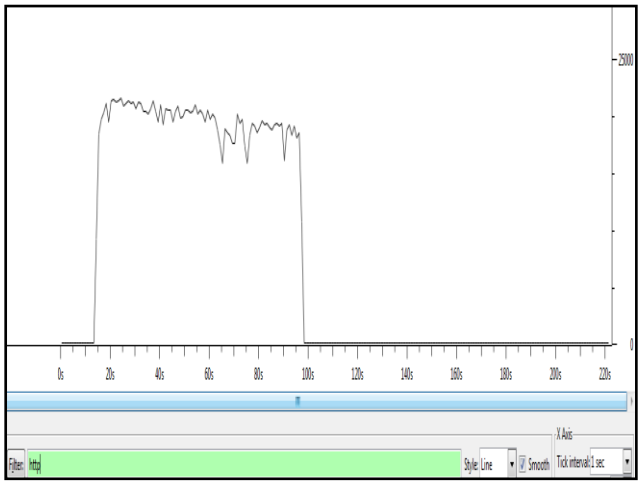

Fig.8 Packet/tick Graph captured from the victim web server interface during the HTTP slow post attack

The victim web server tied up to handle lots of slow httpd processes with receiving large content-length of http header where data's are sent slowly one byte per regular interval. Therefore, it shows the connection was reset to the legitimate users during the attack.

4. PROPOSED MITIGATION MECHANISM

In this paper, the authors approach some mechanism that can protect a web server from application-level DoS attacks, especially, the attacks targeting the resources, including CPU, sockets, or memory of the web server. Since almost all the DoS attack tools intend to disable the server or degrade the performance of the server by offering excessive work to the server or holding the limited resource of the server, authors attempt to detect the malicious node based on the amount of work given by each source node.

4.1 Approach 1: Reverse Web Proxy

A reverse proxy is a kind of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client as though they originated from the server itself or servers themselves. A reverse proxy takes requests from the Internet and forwarding them to servers in an internal network. Those making requests connect to the proxy and may not be aware of the internal network. Reverse proxies can hide the existence and characteristics of an origin server or servers [17]. There are several application firewall features in reverse proxy that can protect against common web-based attacks. Without a reverse proxy, removing malware or initiating takedowns can become difficult. A reverse proxy can distribute the load from incoming requests to several servers, with each server serving its own application area. In the case of reverse proxying in the neighborhoods of web servers, the reverse proxy may have to rewrite the URL in each incoming request in order to match the relevant internal location of the requested resource. A reverse proxy is able to reduce load on its origin servers by caching static content, as well as dynamic content known as web acceleration.

In this paper author proposed an open source web proxy for the mitigation DDoS attack called NGINX [18]. It is a load balancing and as a proxy solution to run services from inside those machines through your host's single public IP address. There is some reason for choosing NGINX likes event driven; its notifications or signals are used to mark the initiation or completion of a process. Thus, the resources can be used by another process until a process initiation event is triggered and resource can be allocated and released dynamically. This leads to the optimized use of memory and CPU. Another one is asynchronous; threads can be executed concurrently without blocking each other. It enhances the sharing of resources without being dedicated and blocked. It also has single threaded; where multiple clients can be handled by a single worker process as the resources are not blocked. Most powerful tools are Nginx DDoS Plugins. Nginx available plugins are testcookie-nginx-module and roboo HTTP Robot mitigation.

4.2 Proposed Solutions Model

In the experimentation, the authors employed a virtual environment setup shown in Figure 9. The hardware specifications, for all servers are same Inter Core i5 processor, 8GB RAM. For web server and reverse web server, authors use centos kernel and for attacker server used backtrack operating system. In this study, the authors specifically focused on the variations of application layer DoS attack: low-rate, slow send and slow read. Both low-rate and slow send require a knowledge of the web server's request timeout to determine the instants at which attack requests should be sent.

In step 2, attacker server attacks on apache server directly server resource optimization becomes saluting condition and it goes down. In step 3, changed the design put a reverse proxy server before on apache server and all request forward to reverse proxy server then apache server. The author measures the web server performance and traffic comes to normal.

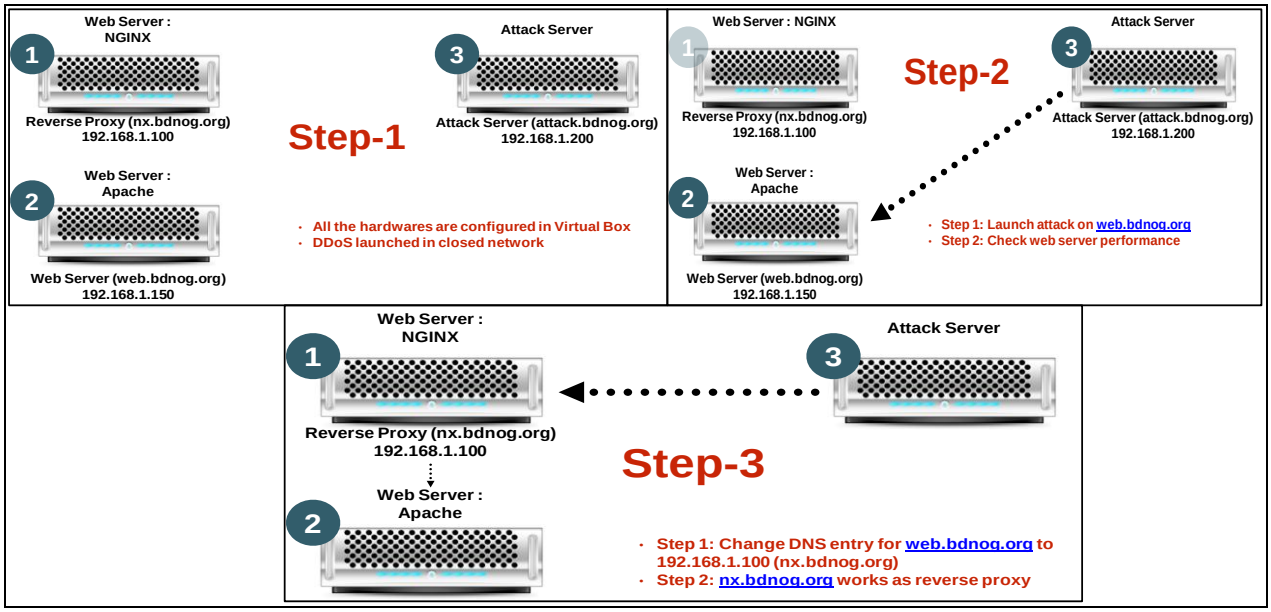

Fig. 9 Proposed Solution Design

4.3 NGNIX

NGINX work processing algorithm show in Figure 10. A dynamic firewall call test-cookies-module is conscientious for deciding if the requests to a particular situation would overload the web application and creating rules to identify and handle these requests by decision engine and reverse proxy processing incoming traffic in accordance with set rules.

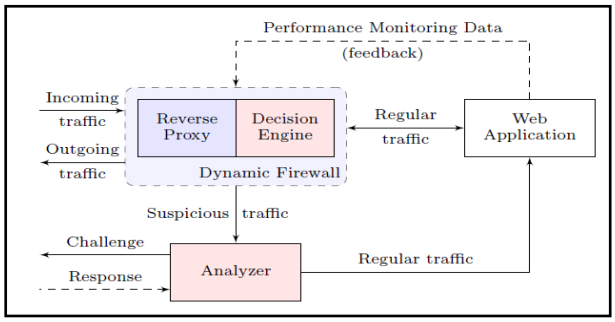

Fig.10 Working algorithm of reverse proxy NGINX

The reverse proxy is a simple context aware http request router, which redirects legitimate requests to the web application and suspicious requests to the analyzer. Reverse proxy routing is rule based; the same philosophy as a regular firewall. The decision engine implements an adaptive system, using test cookies module. When NGINX proxies a request, it sends the request to a specified proxy server, fetches the response, and sends it back to the client. To pass a request to an HTTP proxy server, the proxy pass directive is specified inside a location.

4.4 Scripts (Finding the BOT)

IP set is an extremely useful plugin to iptables, particularly if anyone wants to have a firewall rule that matches against a large set of addresses, ports, or if wants to dynamically change the addresses and/or ports that a rule matches against. The fact is that if anyone use more hosts 1000 through iptables average load becomes huge, since iptables not working with a large number of hosts. Some scripts are given here to find the BOT from the web server easily [19]. In Figure 11 (a) shown finding GET requests from access log file. In Figure 11 (b) finding all access IP address logs from the web server. In Figure 11 (c) short most possible BOT IP address which may attack on the web server. In Figure 11 (d) creating a blacklist in server and set all IP address found in Figure 11(c) as a blacklisted.

```
[root@web ~]# more /var/log/apache2/access.log | grep "bdnog\.org" | grep "GET / HTTP"

203.188.170.218 - - [26/Oct/2013:17:58:25 +0600] "GET / HTTP/1.1" 200 537"http://
www.bdnog.org/site/index.php?url=www.bdnog.org" "Mozilla/5.0 (compatible; MSIE 9.0; Windows
NT 6.1; Trident/5.0)"
94.109.96.192 - - [26/Oct/2013:17:58:26 +0600] "GET / HTTP/1.1" 200 537 "http://
www.bdnog.org/" "Mozilla/5.0 (Linux; U; Android 2.3.4; en-gb; SonyEricssonWT19iv Build/
4.0.2.A.0.58) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"
203.188.170.218 - - [26/Oct/2013:17:58:26 +0600] "GET / HTTP/1.1" 200 537 "http://
www.bdnog.org/site/index.php?url=www.bdnog.org" "Mozilla/5.0 (compatible; MSIE 9.0; Windows
NT 6.1; Trident/5.0)""
```

Fig.11 (a) Finding the GET requests from the server

```
[root@web ~]# more /var/log/apache2/access.log | grep "bdnog\.org" | grep "GET /
HTTP" | cut -d " " -f1

94.109.96.192
203.188.170.218
94.109.96.192
203.188.170.218
```

Fig.11 (b) Finding the GET request IP address from the server

```
[root@web ~]#  more /var/log/apache2/access.log | grep "bdnog\.org" | grep "GET / HTTP" | cut -d " "
-f1 | sort | uniq -c| awk '{if($1>K){print $2}}'
Replace K with value

114.130.136.182
117.18.229.59
117.18.231.60
175.140.219.213
202.134.10.135
```

Fig.11 (c) Sorting the GET request IP address from the server

```
[root@web ~]# ipset create blacklist hash:net
[root@web ~]# more /var/log/apache2/access.log | grep "bdnog\.org" | grep "GET / HTTP" | cut -d " " -f1 |
sort | uniq -c| awk '{if($1>100){print $2}}' | xargs -tl -I _ ipset -A blacklist _
ipset -A blacklist 114.130.136.182
ipset -A blacklist 117.18.229.59
ipset -A blacklist 117.18.231.60
ipset -A blacklist 175.140.219.213
ipset -A blacklist 202.134.10.135
```

Fig.11 (d) Set all IP address in IPtables

```
[root@web ~]# vim ipset
#!/bin/sh
ipset list
Name: blacklist
Type: hash:net
Header: family inet hashsize 1024 maxelem 65536
Size in memory: 16984
References: 0
Members:
114.130.136.182
203.188.170.218
202.134.10.135
37.160.132.237
```

Fig.11 (e) Script for ipset

```
[root@web ~]#  /sbin/iptables -X DDOS_HTTP_FILTER
[root@web ~]# /sbin/iptables -N DDOS_HTTP_FILTER
[root@web ~]# /sbin/iptables -A DDOS_HTTP_FILTER -p tcp --syn --dport 80 -m set --match-set blacklist
src -j DROP
```

Fig.11 (f) Filter attack IP address in Filter rules

In Figure 11 (e) showed members of blacklisted IP address. In Figure 11(f) filters all blacklisted IP address. Once an IP comes in the blacklist, this IP will be drop automatically in next time until it moves from the blacklist. Now run this command as a frequent interval with the help of schedule Job in linux based platform.

*/1 * * * * root /sbin/iptables/DDOS_HTTP_FILTER

4.5 Evaluation of the proposed DDoS defense mechanism via simulation

In this subsection, the authors evaluate the performance of the proposed DoS defense mechanism via real test bed simulation. For this simulation authors used linux platform. The authors investigate the efficacy of the proposed defense mechanism in protecting web servers from low-rate but resource-consuming attacks, by measuring the server response time with and without the defense mechanism. Figure 20 shows the network topology for simulation. In the simulation model, there are three servers which are on linux platform. One server is attacker server, from where attacker generates the DDoS attack. Another server is a web server with apache. Rest server is our proposed reverse proxy NGINX server.

All the link rates are fixed to 100 Mbps, and the propagation delay on each hop is fixed to 0.25 msec. Authors use the same traffic model for both normal flows and attack flows to investigate the scheme when the attack traffic pattern is indistinguishable from the normal traffic pattern. Each normal client or attack node makes only one TCP connection to an internal server, and sends http request packets in a persistent mode without closing the TCP connection. In order to consider the worst case scenario, authors let each newly established session persist until the end of the simulation. Normal sessions arrive from the beginning of the simulation with an average inter-arrival time of 1 sec. Attack flows arrive after 1000 sec from the start of the simulation, with an average inter-arrival time of 0.5 sec. The http request packets are sent to the server with an exponentially distributed inter-arrival time within each session. The average inter-arrival time of request packets is set to 1 sec for both normal flows and malicious flows. However, authors assume that the request packets of the attackers require more processing time at the server. Thus, the processing time of request packets from normal nodes is modeled by an exponential distribution with an average of 5 msec, and that of request packets from malicious nodes is exponentially distributed with an average of 250 msec.

After applying the defense mechanism authors showed the http incoming traffic in Wireshark. As the tool sent http headers with large content-length lots of traffic of http packets has been sent to the victim web server, however this tool sent one byte of POST data at regular time interval configured by the attacker. During this attack packet vs tick (X-axis shows Time, Y-axis shows packet) graph shows high traffic that captured from the victim server Ethernet interface as like as Figure 8. Now, authors compare the Figure 12 and Figure 8. In Figure 8, it showed that without any defense mechanism we received huge http traffic (25000 packets). Which makes

server resource saturation and out of service. After applying our proposed defense mechanism, again authors start a DDoS attack on the target server. But in this time server showed normal traffic and it's operated fine. In Figure 12 showed http traffic after applying defense mechanism.

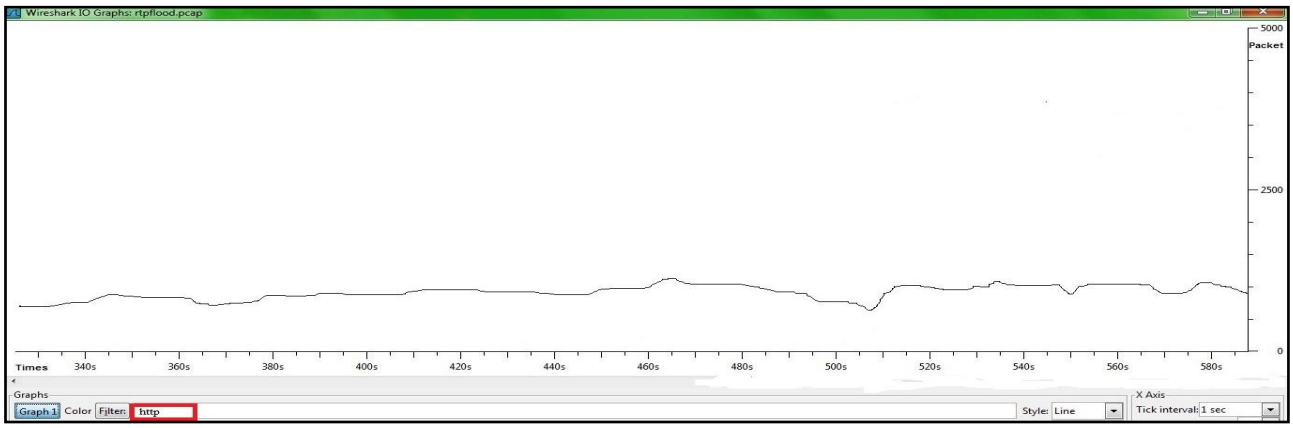

Fig.12 Packet/tick Graph captured from the victim web server interface during the HTTP slow post attack

5. CONCLUSION

In this paper, the authors investigated a new two-stage mechanism that can protect web servers from low rate resource-consuming DoS attacks. The proposed mechanism is based on two key ideas. The first one is a reverse proxy-based admission control scheme in the first stage, which protects the servers from a sudden surge of attack flows. The authors also investigated the condition to detect the victim servers and freeze the whitelist based on the server response time in detail. The second key idea is to detect attack flows based on the concept of a whitelist-based admission control defined for each pair of client and server IP addresses in the second stage.

The experiment results show that the reverse proxy based scheme protects the server at the initial stage of DDoS attack, and the whitelist-based admission control policies attack flow detection mechanism distinguishes attack flows from normal flows and effectively filters the IP addresses of the attackers from the banned list. Although authors focused on protecting http-based web servers in this paper, the proposed approach will be extended to other types of web servers in future study.

6. REFERENCES

[1] D. Dagon, G. Gu, C. P. Lee, W. Lee, "A Taxonomy of Botnet Structures," in Proc. of Annual Computer Security Applications Conference (ACSAC), Dec. 2007.

[2] www.arbornetworks.com

[3] T. Peng, C. Leckie, K. Ramamohanarao, "Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems," ACM Computing Surveys, vol. 39, no. 1, pp. 1-42, Apr. 2007.

[4] S. Kandula, D. Katabi, M. Jacob, A. W. Berger, "Botz-4-sale: surviving organized DDoS attacks that mimic flash crowds," in Proc. of NSDI, Boston, MA, 2005.

[5] C. Estan, G. Varghese, "New Directions in Traffic Measurement and Accounting," in Proc. of ACM SIGCOMM, Aug. 2002.

[6] R.R. Kompella, S. Singh, G. Varghese, "On Scalable Attack Detection in the Network," in Proc. of ACM Internet Measurement Conference (IMC), Oct. 2004.

[7] Jose Nazario, BlackEnergy DDoS Bot Analysis, Technical report, Arbor Networks, Oct. 2007.

[8] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, K. Han, "Botnet Research Survey," in Proc. of IEEE COMPSAC, pp. 967-972, 2008.

[9] ha.cker.org security lab, Slowloris HTTP DoS, http://ha.cker s.org/slowloris/

[10] J. Mirkovic, P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," SIGCOMM Computer Communication Review, vol. 34, no. 2, pp. 39-53, 2004.

[11] A. Kuzmanovic, E. Knightly, "Low-rate TCP-targeted denial of service attacks (the shrew vs. the mice and elephants)," in Proc. of ACM SIGCOMM, pp. 75-86, 2003.

[12] G.Macia-Fernandez, J.E.Diaz-Verdejo, P.Garcia-Teodoro, "Evaulation of a low-rate DoS attack against application servers," Computers & Security, vol. 27, no. 7, pp. 335-354, 2009.

[13] H. Sun, J. Lui, D. Yau, "Defending against low-rate TCP attacks: dynamic detection and protection," in Proc. of 12th IEEE International Conference on Network Protocols (ICNP04), pp. 196-205, 2004.

[14] W. Wei, Y. Dong, D. Lu, G. Jin, H. Lao, "A novel mechanism to defend against low-rate denial-of-service attacks," Lecture Notes Comput. Sci. 3975,pp. 261-271, 2006.

[15] G. Macia-Fernandez, R. A. Rodriguez-Gomez, J. E. Diaz-Verdejo, "Defense techniques for low-rate DoS attacks against application servers," Computer Networks, vol. 54, no. 15, pp. 2711-2727, 2010.

[16] M Srivatsa, A. Iyengar, J. Yin, "Mitigating application-level denial of service attacks on web servers: a client-transparent approach," ACM Transactions on the Web, vol. 2, no. 3, pp. 15:1-15:49, July 2008.

[17] http://en.wikipedia.org/wiki/Reverse_proxy

[18] www.nginx.com

[19] Mohammad Fakrul Alam, "Application Layer DDoS, A Practical Approach & Mitigation Techniques, "South Asian network Operators Group (SANOG) -23 Conference, Thimpu, Bhutan, 2014.