Journal of Robotics and Control (JRC)
Volume 1, Issue 4, July 2020
ISSN: 2715-5072, DOI: 10.18196/jrc.1427

Mitigating Denial of Service Attacks with Load Balancing

Adaoma Ezenwe[1], Eoghan Furey[2], Kevin Curran[3]
[1, 2] School of Computing, Letterkenny Institute of Technology, Co. Donegal, Ireland
[3] School of Computing, Engineering & Intelligent Systems, Ulster University, Northern Ireland
eoghan.furey@lyit.ie[1,2], kj.curran@ulster.ac.uk[3]

***Abstract*—Denial of service (DOS) attacks pose a tremendous risk to organizations. The attacks have shifted from attacks at the layer 3 and layer 4 (network level) to layer 7 attacks, which are not quickly detectable by firewalls and utmost Intrusion Detection systems. An accelerating number of those attacks against the infrastructures of web servers of numerous organizations has been reported. The research aims to investigate some layer 7 application DOS attack tools and test open-source tools that offer some defense against these attacks. The research used open-source load balancing software, namely HAProxy as the front line of defense against DOS attacks to assess the effectiveness in detecting and preventing layer 7 DoS attacks. We demonstrate how a properly configured HAProxy can handle a variety of DOS attacks in a much more efficient manner.**

Keywords—DOS, attack, layer, load balancing HAProxy

I. INTRODUCTION

Denial of service attacks is a major threat to online business infrastructures. The attack intended to shut down the machine or cut off network connection. Symantec recorded a vast number of online business attacks [1]. The attacks move from the network layer, which requires massive resources and great bandwidth to perform. An attacker operating from a single appliance will submit massive traffic to overwhelm a web server which makes the system inaccessible for other authentic users. Most businesses have primarily relied on e-commerce for the majority of their revenue and service provision. The attacks continue to be a major threat to the network of online business.

Some companies rely on social media sites for advertising and direct customer contact. A reduction of quality of the service will cost money to some businesses [3]. The danger of DoS attacks on online companies is rising equally. In 2016, the largest DOS attack scale ever reported, was performed on DNS provider, Dyn, utilizing botnets infected with Mirai's Internet of Things (IoT). The attack hit major websites including PayPal, Facebook and Netflix. Sometimes, such attacks are utilized as distracting tactics to distract concentration from other cyber-attacks, which contribute to the data-stealing. Verisign reported a 75 % increase in the pattern of Layer 7 DOS attacks across all verticals of business [4]. Given the growing prevalence of the attacks, the work seeks to use open source software as well as tries to explore how resources can support to detect and deter a DoS attack as well as allowing post-event threat analysis. An HAProxy is an open-source platform for load balancing and defense mechanisms against DOS attack from Layer 7. Two attacks, namely the HTTP GET and HTTP POST are common as they manipulate a function of the HTTP protocol that leave the connection open while waiting for the web server receives the legitimate HTTP full request [5].

Attackers take advantage of such a vulnerability to block system resources that other legitimate web application consumers are denied access to the client because threads, full link sockets, and other device resources are left open to unscrupulous attackers. The researchers are trying to see whether HAProxy can be used to protect against a DOS attack and discuss the potential of the open-source platform, ELK Stack, in providing real-time data log reviews as well as warning approaches when a server is going down or when an attack is going on by monitoring device and logging information in near real-time.

Denial of service attacks (DOS) is where infrastructure and services for legitimate users are rendered. The DDOS (Distributed Denial of Service) attack is an enhanced variant of DOS, where several infected devices are hired for the attack [6]. The DDOS is a sophisticated type of DOS, where bots comprising corrupted networks or computers, are involved instead of a person or a specific device perpetrating the attack. These bots or Botnets are set up by leveraging the inherent weaknesses of computer systems and operating such systems in a coordinated broad-scale attack on target systems [7]. These attacks have potential consequences such as service interruption, network unavailability, and may result in data failure being triggered to user device resources [8]. Distributed DOS attacks have lately become one of the biggest threats to the stability and security of the networks and web services [9]. These attacks involve the attacker's immense bandwidth and resource that is challenging to arrange, excepting the case of an individual's HTTP flood attack or using a tiny botnet [10]. Another approach would be to identify the attacks based on the targeting of the network layer, primarily the Network/Transport layers as well as the application layers. The classification of the attacks can be based on the attack's resource target, primarily Bandwidth Usage and Use of Server Resources [11]. One means of categorizing DOS attacks is through the mode of attack either reflector/amplification attack or direct attack [12].

The seventh layer of the OSI model is the application layer. It concerns how the user primarily communicates with the HTTP(S), SMTP, FTP, a protocol stack, etc. [13]. Because of the existence of the layer that allows direct user interaction, attackers can bring down a website by sending repeated queries, continual reloading and demanding details from the database that may or may not exist. DOS attacks against web applications have been becoming more common with layer 7 GETs and POST attacks amongst the latest techniques of DOS attacks [8] claiming to be very complicated methods for DOS attacks at the application layer. Standard DOS identification and prevention methods are ineffective for these Layer 7 attacks [12] as in application layer DOS attacks. Compromised devices submit high volume pernicious database service requests via regular TCP connections to the target web servers [9]. The attacker aims to constantly requesting a heavy URL from the victim in order to drain the target's computing resources. This layer attacks are typically more complex and have more damage on the victim's computing capital while costing very little money to the attacker to perpetrate the attack [14].

The HTTP Post attack is powerful and effective, occurring when an intruder infiltrates the webserver with gradually abounding data inside the request body of the HTTP Post. Because it is an HTTP protocol-compliant request, it has to keep the connection open while the attacker drains the resources, leaving the web application unavailable to legitimate site users [14], [8]. The attacker sends the request header rapidly and defines the size of the message body that the web application will anticipate. The intruder subsequently sends the full HTTP header info allowing the contact appears quite valid but later sends the message body thousands of already established connections at a rate of one byte per 100 seconds. The intruder makes the request-transfer rate incredibly slow, and this hogs the web-server's memory power and CPU, irrespective of the hardware capacity of the web-servers, as it awaits complete data transmission before the link is closed [15]. HTTP GET queries occur when a user enters a URL in the web-browser's address bar and clicks the enter button or when a web-application user seeks hyperlinks in a web application.

The HTTP Get attack imitates the behavior of website's genuine users, but the intruder sends an abundant HTTP Get requests by means of botnets or other methods to make the application inaccessible to legitimate users. Such requests are usually identical with the standard HTTP request-bar. It is difficult to distinguish such attacks because the requests are typically sent through valid network packages, standard TCP link, and request the web legitimate content [16]. Popular defense mechanisms detecting and filtering DOS traffic on the basis of the illegitimacy of the request and the request rate are ineffective in mitigating this form of attack [17], [18]. This is because "the traffic to the attacker is as legitimate as the traffic to normal users"[19]. A sample of such attacks is HTTP GET slow read request attack, commonly referred to as the Slow READ attack. The attacker continually hits the web servers with requests, which leaves the connection open, resulting in the waste resources-memory of the client, CPU time [10].

Prevalent load balancers like F5 and Cisco may be utilized to protect against HTTP GET DOS attacks [20], [21]. A load balancer accommodating a large number of requests and communications from a large number of users and devices is important [22]. HAProxy standing for the High Availability Proxy is a prevalent HTTP load balancer and open source TCP [23]. It provides several solutions for load balancing algorithms and conducts back-end server health checks before it routes traffic to only stable nodes [23]. ackend server status and incoming and outgoing traffic detai ls may be evaluated using the Details method (Figure 1).
HAProxy creates logs that can help uncover which program causes a problem in one event. It may operate in TCP mode, testing whether databases such as MySQL server are running and in HTTP mode.

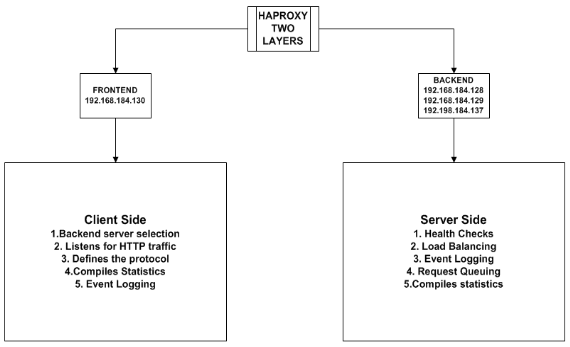

Fig. 1. Frontend and Backend layers of HAProxy

The backend of the HAProxy consists of one or several servers, to which requests submitted are delivered. Backend setup consists primarily of web servers, HAProxy ports for listening, a health check system, a load balancing algorithm, and several other specialized configurations. The frontend specifies the front end's port number and IP address, some ACLs and other specialized settings.

High Availability is the capacity of a program striving to achieve a more advanced output including uptime. Such function uses an active failover program which cuts downtime for a business [24]. A high-availability system ensures that errors are found when arising and measures are set to minimize those errors, and confirms that there is no single point failure in the entire system so that the user never suffers down-times. There are various formulas that calculate the amount of downtime and each company determines what degree of downtime is appropriate on the basis of service level agreements (SLAs). Most systems have security susceptibilities that hackers may leverage to trigger denial of service functionality for legitimate device customers, and sometimes result in server downtime. In a production environment, a simple advantage of HAProxy provides users with a high availability when correctly installed, and backend nodes are allowed to be withdrawn or introduced to the HAProxy with no down-time [23]. Failover is the system's ability to stay working while one or more of its elements are down.

HAProxy gives the capability to designate web servers as backups, and when main web-servers go down, the backups are restored into an active pool. By announcing the option "all backup in configuration file". The backup server farms may be linked to the active HAProxy pool [25]. This server is designed to be failover farm when all servers are down. When all active and backup services are inaccessible, it is important to use a personalized error page to inform consumers on efforts to restore service [26]. HAProxy facilitates load balancing algorithms like Round Robin, and more complicated algorithms like Least Link Origin, IP Hash, URL Hash, and Weighted-RR.

ELK Stack is a grouping of three open-source resources, namely Elasticsearch, Logstash, and Kibana. Elasticsearch is Apache Lucene project-based text indexing and search engine application for profound searches, queries, and data analysis [27], [28]. Kibana is the ELK Stack's visualization portion. It offers graphical displays of data retrieved from the Elasticsearch and offers the ability to easily view vital info probably mix up with big data.

II. Evaluation

The attacker connects to the HAProxy server’s IP address through the' internet'. The HAProxy server collects traffic via its frontend and afterwards allocates the traffic to the web server’s backend. The logs stored on the HAProxy server is sent by means of Beats log shipping tools, namely Metricbeat and Filebeat to ELK server. Figure 2 illustrates the system architecture.

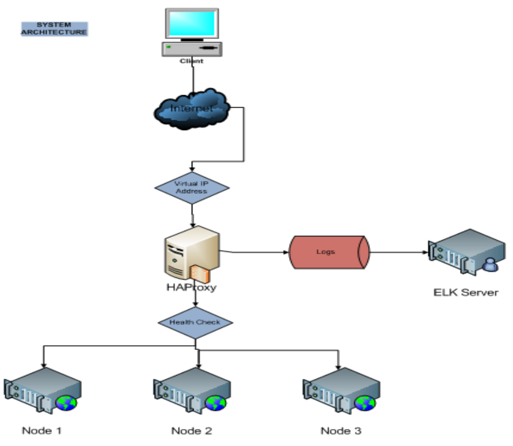

Fig. 2. System architecture

HAProxy is installed on a server that handles data from web users and loads the data as a first line of defense against DOS attack in order that unexpected large surges in HTTP traffic flows do not overload the backend servers. The HAProxy server consists of a frontend in which all access to the backend servers is reached first, and the backend HAProxy service is connected to the backend web servers later. A load balancing helps remove single point of failure in a system to help recognize errors and differentiate malfunction component faults. The HAproxy load balancer allocates the traffic to the backend servers according to the designed load balancing formula, and it eliminates the simple bottlenecks in the HTTP traffic. To assist with replication, HAProxy enables server setup as an active-backup (failover) in the case of one or more of the main servers crashing. The configuration is designed where the administrator considers it crucial to put a another server into the load-balanced servers active pool to deliver another layer of DoS attack protection once the main servers have been marked down. The Elasticsearch, Logstash, and Kibana, installed on the ELK server are utilized for distributed storage, data processing, log review, as well as log virtualization. To create DOS attack, GoldenEye DoS attack tool is used, and the attack traffic is forwarded from the HAProxy server to the ELK server for review. The HAProxy statistics page gathers and analyzes the event logs created throughout the DOS attack to identify patterns that can be used when the alarm is activated when such an event takes place. The HAproxy service collects data, utilizing the required load balancing algorithm to disperse the traffic to backend servers.

To evaluate the effects of the DOS attack and build a test for evaluating the efficacy of the load balancer and its security measures, the DOS attack was experimented on two web servers operating the Apache to see whether the result is equivalent on both. Similar experiments were performed on the server operating a standard setup to monitor the results of the attack without having to implement extra security mechanism to help mitigate the DOS threat. Lastly, the HAProxy server is safe for DOS, and the same experiment is undertaken to seeing the efficacy of DOS security settings to prevent DOS attacks. Using the GoldenEye DOS attack method, HTTP traffic was sent to web servers through the server’s IP address to generate adequate traffic to analyze the log.

SlowHTTPTest was used to test a Denial of Service Attack at Layer 7 implemented on the Slowloris attack, Slow POST attack, Slow READ attack, and Apache Range Header attack. The attack length was 240 seconds by default, and the connections listed is 1000 with 200 connections per-second for all the tests. It used default content header length of 4096. The timeout probe was 3 seconds with a 10-second period between launch of the attacks. The attack tool determined the follow-up data size for the HTTP POST attack, slow BODY attack, and the Extra data max length in the case of Slow HTTP headers attack based on the default configuration.

Experiment 1: Attack directly on Web server using slowhttptest

Experiment 1 aimed to determine what effect it would have to submit massive number of queries directly to a web server without using a load balancer and to evaluate if the balancer offers some level of security toward the DoS attack relative to the launching of the attack. The experiments conducted on both web servers without using a load balancer displayed comparable results. The web servers did not recover from the Slowloris attack until the maximum attack time limit 240 seconds.

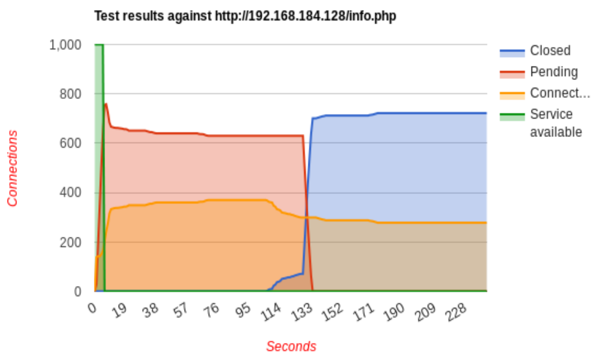

Fig. 3. Graph showing the status of the connection

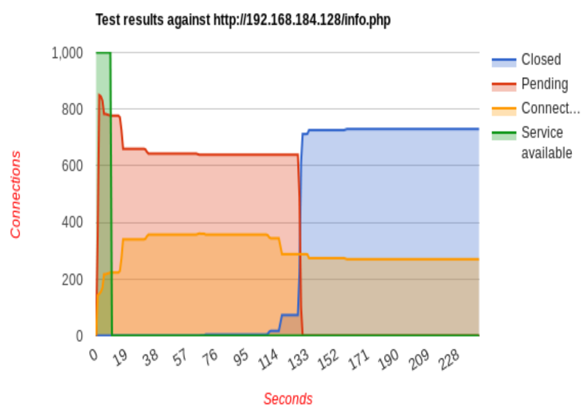

Fig. 4. Graph slowing the READ attack directly on the web server

We performed a slow BODY/slow POST attack against the web servers using a fake HTTP request term, CHECKVERB, and parameter to the targeted web server, VM1 while also encountering the attack on the web servers. The attack sent a 10000 content-length values but sent out 22 per 110 seconds. It made busy web server resources. After the 5th second, the webserver was unable to provide any service, and the system finally crashed at the 110th second, finish the attack with Connection's exit status-declined (see Figure 3). The final experiment was the sluggish READ attack aimed at the webserver. The attack enabled 1000 valid HTTP requests per second and a 10 to 20-second receive window range. It read the response at around 6 to 7 bytes per second.

Figure 4 displays the specifics of the test parameters applied during the attack and shows the status representation of the 1000 connections over the course of the attack. The web server attached for 223 requests but kept pending the remining 777 requests. Service became inaccessible at the 10th second and during the subsequent 230 seconds, 270 connections were enabled. A total of 730 connections were closed at the 240th second. The service became inaccessible during the 10th second and the remaining seconds of the attack.

Experiment 2: Attack on Web Servers via Basic HAProxy Server

Experiment 2 was carried out using load balancer but rendered without any DOS protection configuration settings. The first test applied on the unsecured HAProxy server was the Slow Header/Slowloris attack. The same research criteria were used in the study. The HAProxy server recorded time-length of the attack, 240 seconds. Figure 5 displays the parameters and the graphical depiction of the attack over the entire time of the experiment. To obtain a benchmark to evaluate the efficacy of the HAProxy's security settings in preventing the Slowloris attack, the number of the connections was increased to perceive its impact on the HAProxy server. The service identified by HAProxy attack stopped working at the 10th second but recovered att the 15th second. After the attack stopped, 1664 connections had been successful while 336 connections were closed. Figure 6 shows the connection status till the time limit of the attack.

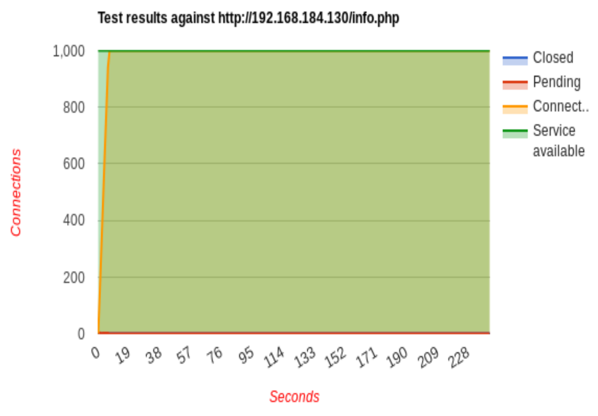

Fig. 5. Graph displaying the connections for the slow header attack on the HAProxy server

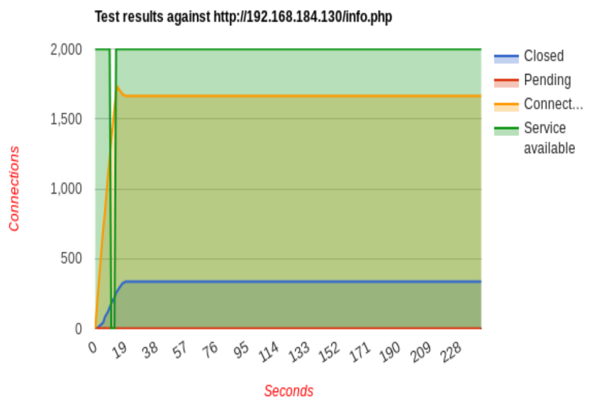

Fig. 6. Graph displaying the result of 240s of Slowloris attack on the unsecured HAProxy

A fake Body/Slow POST attack was performed using a fake HTTP verb (CHECKVERB). The service became inaccessible at the 10th second but recovered at the 55th second. Nonetheless, the server was unable to take any new connections at 70th second. The load balancer recorded a status of No Open Connection Left though the available service and no web servers crashed due to the overload. Graphical illustration of the connection status through the course of the attack was displayed in Figure 7.

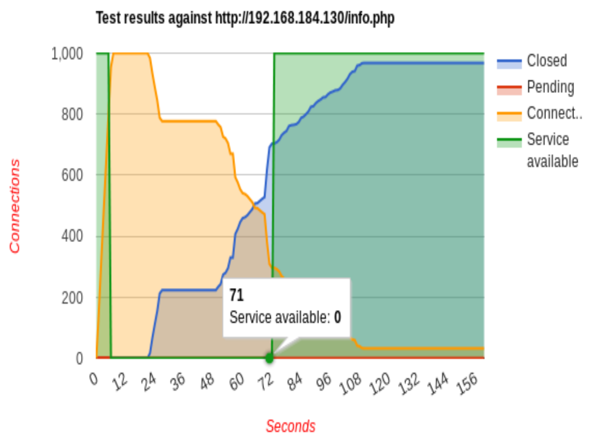

Fig. 7. Graph displaying Slow body attack on the unsecured HAProxy

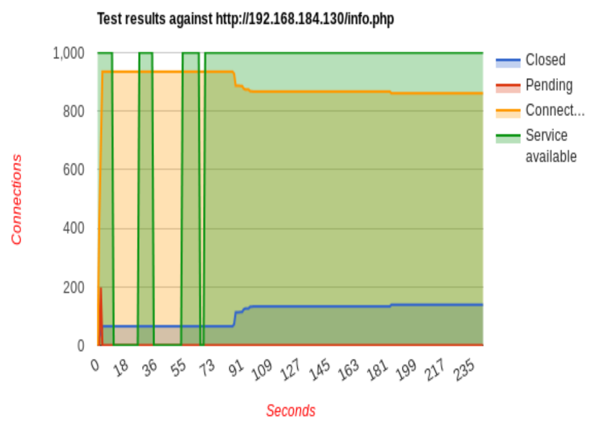

Fig. 8. Graph of the service availability for the duration of the attack

The final test on the unsecured HAProxy server is the slow READ attack. The number of receiving windows in the attack was 10 -20, and the read rate was 6 to 7 bytes per second. After 5 seconds of testing, if the server is active, the attack device re-coordinates and relaunches the attack. The HAProxy recorded service inaccessible at the 10th second but retrieved at the 30th second. Nevertheless, the retrieval did not last because the service fluctuated several times within the first 80 seconds. However, it was restored at the rest of attack time-duration.

To restore the service, the system was rebooted. The HAProxy with the basic configuration showed high flexibility during the Slowloris attack relative to slow POST and slow READ attack. Service availability of the Slowloris attack was 100% when 1000 connections were used per second, but when the connection rate per second was doubled, 17% of the connections were decreased. Nevertheless, for 95% of the attack time-period, the service was constant. With the default setup, the HAProxy server was unable to provide service for about 35% of the attack time-period. Approximately 830 connections were retained while the remaining 170 connections were discarded. The worst result was for the slow POST attack. The service was inaccessible for 45 seconds out of the 70 seconds. Yet, all accessible connections were depleted at the 70th second.

The result of the experiment found that web servers could only sustain the service for less than a fixed time of 10 seconds when directly attacked. A server crash was reported while the launch of attack on the web server. The HAProxy showed better handling for the Slowloris attack compared to the other attacks.

Experiment 3: Attack on Web Servers Via Secure HAProxy Server

The HTTP-request timeout was set for five seconds to a suitable number to avoid the Slowloris attack. After applying the security settings to the HAProxy config file, we carried out the Slowloris attack. The corresponding Attack Order was released on the secured HAProxy server. The attack persisted within 10 seconds. The test finished on 11th second with a status of no open connection remaining, but the service was reported available. The warning of no available connections remaining was validated to apply only to the attacker's connections. Separate connections made from another device suggested that connections and the service were still accessible. Figure 9 displays that all connections after five seconds were closed because the config file was set up with a connection timeout of five-second if a full request is not received.

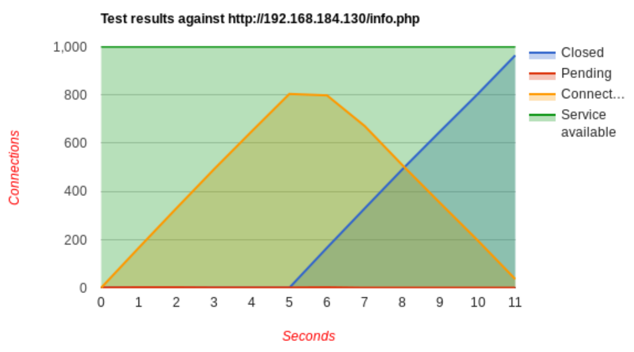

Fig. 9. Graph shows that all connections after 5 seconds were closed

Later, we carried out a Slow BODY attack and Slow READ attack, launched on the secure HAProxy server. The Slow BODY attack lasted for about 10 seconds. The server recorded no open connections remaining at the 10th second of attack. The service was only inaccessible to the attacker but still accessible for all other users. The Slow READ attack sent a 10 to 20 receive window but read at the rate of 6 to 7 bytes per second. The attack continued for 240 seconds, and the service was inaccessible for merely five seconds. The HAProxy server closed 980 connections requests on average and connected 20. During the attack, the connection cut after the 35th second but recovered at the 40th second. It was similar to the previous attack. The secure HAProxy server was capable of handling the Slowloris attack. The attacker's connections were effectively restricted to the limits configured in the configuration file. All connections to the ongoing attack were closed.

III. CONCLUSION

This paper delved into layer 7 attacks with a focus on slow BODY, slow POST, and slow READ attack techniques. When a web application traffic is directly redirected to a web server, it exposes a single point of web infrastructure's failure and is not a sensible method given the current patterns, in which daily continuous traffic to the web applications has become so critical. The reasoning behind all these methods is the use of modules integrating dynamic-scheduling mechanism to dynamically assign backend servers to handle incoming requests in the list of queues with various priority levels. This method proved ideal both for load balancing and for avoiding DOS attacks. Such a method has proven to be proper for load balancing as well as for preventing DOS attacks.

The improvement of the HAProxy's security has created a significant performance enhancement. Connection to a web application is denied to an attacker based on the traffic patterns which he presents. A usual web application user does not open 2000 connections in a second and 200 simultaneous connections. HAProxy senses this feature, and then prevents the attacker. The implementation of the time-out of an HTTP request and putting buffer size's limitation stops the slow POST attack from consuming backend web server resources.

It is reasonable to configuring a free open source and a lightweight load balancing algorithm to improve performance, high availability, and simultaneously function as the first powerful layer of protection against the DOS attack. Ultimately, using preference manager modules along with several buffer queues with different priorities may assist filtering and sorting incoming requests. according to different priority levels depending on the authenticity of the request's sources or the irregular existence of the request traffic.

References

[1] Symantec, “Internet Security Threat Report (ISTR)”, 2017. https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

[2] Masdari, M.; Jalali, M., “A survey and taxonomy of DDoS attacks in cloud computing”. Secure Communication Networks, Vol. 9, No. 1, pp:3724–3751, SCN-15-0746.R1, 2016.

[3] Vlajic, N. and Slopek, A., “Performance and economies of ‘bot-less’ application-layer DOS attacks.” The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014). 2014, DOI: 10.1109/ICITST.2014.7038828, 8-10 Dec 2014, London, UK,

[4] Verisign, “Distributed Denial of Service Trends Report”. Verisign2017. https://www.verisign.com/en_GB/security-services/DOS-protection/DOS-report/index.xhtml

[5] Bonguet, A., Bellaiche, M., “A Survey of Denial-of-Service and Distributed” Denial of Service Attacks and Defenses in Cloud Computing. Future Internet 2017, 9(3), 43, 2017; doi:10.3390/fi9030043

[6] Santanna, J., Rijswijk-Deij, R., Hofstede, R., Sperotto, A., Wierbosch, M., Granville, L. and Pras, A. “Booters — An analysis of DOS-as-a-service attacks.”, 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). 2015 DOI: 10.1109/INM.2015.7140298, 11-15 May 2015, Ottawa, Canada,

[7] Merouane, M. “An approach for detecting and preventing DDOS attacks in campus.”, Automatic Control and Computer Sciences, January 2017, Volume 51, Issue 1, pp 13–23, 2017

[8] Mahadev, A., Kumar, V. and Kumar, K. “Classification of DDOS Attack Tools and Its Handling Techniques and Strategy at Application Layer.”, International Conference on Advances in Computing, Communication, & Automation (ICACCA), 30 Sep – 1 Oct 2016, 2016, DOI: 10.1109/ICACCAF.2016.7749002,

[9] Sree, R., Bhanu, S. “HADM:detection of HTTP GET flooding attacks by using Analytical hierarchical process and Dempster–Shafer theory with MapReduce. Security and Communications Networks, Vol. 22, No. 1, pp.4341-43574, 2016.

[10] Park, J., Iwai, K., Tanaka, H. and Kurokawa, T. “Analysis of Slow Read DoS Attack and Countermeasures on Web servers.”, International Journal of Cyber-Security and Digital Forensics, Vol. 4, No. 2, pp.339–353, 2015. http://dx.doi.org/10.17781/P001550

[11] Srivastava, R., Verma, A., Hussain, S. An Approach for Load Balancing Among Multi-Agents to Protect Cloud Against DDos Attack.”, International Journal of Computer Trends and Technology (IJCTT), Vol. 9 , No 5, Mar 2014. ISSN: 2231-2803. Pp: 217-228, 2014.

[12] Fouladi, R., Kayatas, C. and Anarim, E. (2016) Frequency based DDOS attack detection approach using naive Bayes classification. 2016 39th International Conference on Telecommunications and Signal Processing (TSP). June 27-29 2016, Vienna, Austria, DOI: 10.1109/TSP.2016.7760838

[13] Modiri, N., “The ISO reference model entities.”, IEEE Network, [online] 5(4), pp.24–33, 1991. DOI: 10.1109/65.93182. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=93182

[14] Hirakawa, T., Ogura, K., Bista, B.B. and Takata, T.,.” A Defense Method against Distributed Slow HTTP DoS Attack.” 2016 19th International Conference on Network-Based Information Systems (NBiS), 2016, 7-9 Sep 2016, Ostrava, Czech Republic, DOI: 10.1109/NBiS.2016.58

[15] OWASP. “Application Denial of Service.” OWASP White Paper, 2010, March 2010 https://www.owasp.org/index.php/Application_Denial_of_Service

[16] Chwalinski, P., Belavkin, R. and Cheng, X, “Detection of application layer DDOS attack with clustering and likelihood analysis.”, ModGlobecom 2013 Workshop - First International Workshop on Security and Privacy in Big Data, Atlanta, USA, 2013 DOI: 10.1109/GLOCOMW.2013.6824989

[17] Oikonomou, G. and Mirkovic, J. “Modelling Human Behaviour for Defense Against Flash-Crowd Attacks.”, 2009 IEEE International Conference on Communications, 2009. DOI: 10.1109/ICC.2009.5199191, 14-18 June 2009, Dresden, Germany, http://ieeexplore.ieee.org/document/5199191

[18] Bekeneva, V., Shipilov, N., Borisenko, K. and Shorov, A. “Simulation of DDOS-attacks and Protection Mechanisms against Them.” Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), 2015 IEEE NW Russia. 2015, Available at: http://ieeexplore.ieee.org/document/7102230/.

[19] Tang, C., Lee, E., Tang, A. and Tao, L. “Mitigating HTTP Flooding Attacks with Meta-data Analysis.” 2015 IEEE 17th International Conference on High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7336365.

[20] Ahmed, A., Sadiq, A. and Zolkipli, M. “Traceback model for identifying sources of distributed attacks in real time.” Security and Communications Networks, Vol. 44, No. 2, pp.2173-2185, 2016..

[21] Kaur, P., Kumar, M. & Bhandari, A. “A review of detection approaches for distributed denial of service attacks,” Systems Science & Control Engineering, Vol. 5, No. 1, pp: 301-320, 2017. doi:10.1080/21642583.2017.1331768

[22] Hou, L., Zhao, S., Xiong, X., Zheng, K., Chatzimisios, P., Hossain, M.S. and Xiang, W. “Internet of Things Cloud: Architecture and Implementation.” IEEE Communications Magazine, Vol. 54, No. 12, December 2016, 2016. DOI: 10.1109/MCOM.2016.1600398CM

[23] Kaushal, V., and Bala,M.. “Autonomic Fault Tolerance Using HAProxy in Cloud Environment.” International journal of advanced engineering sciences and technologies, Vol No. 7, Issue No. 2, pp: 222 – 227, 2010.

[24] Sun, J., Dong, X., Zhang, X., Gong, W. and Wang, Y. “High availability analysis and evaluation of heterogeneous dual computer fault-tolerant system.” 2014 IEEE 5th International Conference on Software Engineering and Service Science, 2014. DOI: 10.1109/ICSESS.2014.6933605, 27-29 June 2014, Beijing, China,

[25] HAProxy, “Failover and worst case management with HAProxy”, 2013. https://www.haproxy.com/blog/failover-and-worst-case-management-with-haproxy/

[26] Lassnig, M., Vigne, R., Beermann, T., Barisits, M., Garonne, V. and Serfon, C. “Scalable and fail-safe deployment of the ATLAS Distributed Data Management system Rucio.” Journal of Physics: Conference Series, 12 May 2015, pp: 140-148, 2015. https://cds.cern.ch/record/2015469/files/ATL-SOFT-PROC-2015-023.pdf

[27] Oliverira, R., Ferreira, D., Ferreira, R., Cruz-Correia, R. “Open-Source Based Integration Solution for Hospitals. Computer-Based Medical Systems (CBMS),” 2016 IEEE 29th International Symposium, Open-Source Based Integration Solution for Hospitals, 2016. DOI: 10.1109/CBMS.2016.44,

[28] Smith, G. “Log Analysis with the ELK Stack.” LinuxFest North West Blog. October 11 2016, 2016. https://www.linuxfestnorthwest.org/sites/default/files/slides/Log%20Analysis%20with%20the%20ELK%20Stack.pdf.