I. J. Computer Network and Information Security, 2012, 12, 31-39
Published Online November 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2012.12.03

Threshold Based Kernel Level HTTP Filter (TBHF) for DDoS Mitigation

Mohamed Ibrahim AK and Lijo George
Trichy Engineering College, Tiruchirappalli, Tamil Nadu, India
{ibrahim.awn, lijogeorgemail}@gmail.com

Kritika Govind and S. Selvakumar
National Institute of Technology, Tiruchirappalli, Tamil Nadu, India
{kritika, ssk}@nitt.edu

***Abstract* —** HTTP flooding attack has a unique feature of interrupting application level services rather than depleting the network resources as in any other flooding attacks. Bombarding of HTTP GET requests to a target results in Denial of Service (DoS) of the web server. Usage of shortened Uniform Resource Locator (URL) is one of the best ways to unknowingly trap users for their participation in HTTP GET flooding attack. The existing solutions for HTTP attacks are based on browser level cache maintenance, CAPTCHA technique, and usage of Access Control Lists (ACL). Such techniques fail to prevent dynamic URL based HTTP attacks. To come up with a solution for the prevention of such kind of HTTP flooding attack, a real time HTTP GET flooding attack was generated using d0z-me, a malicious URL shortener tool. When user clicked the shortened URL, it was found that the user intended web page was displayed in the web browser. But simultaneously, an avalanche of HTTP GET requests were generated at the backdrop to the web server based on the scripts downloaded from the attacker. Since HTTP GET request traffic are part of any genuine internet traffic, it becomes difficult for the firewall to detect such kind of attacks. This motivated us to propose a Threshold Based Kernel Level HTTP Filter (TBHF), which would prevent internet users from taking part in such kind of Distributed Denial of Service (DDoS) attacks unknowingly. Windows Filtering Platform (WFP), which is an Application Programming Interface (API), was used to develop TBHF. The proposed solution was tested by installing TBHF on a victim machine and generating the DDoS attack. It was observed that the TBHF completely prevented the user from participating in DDoS attack by filtering out the malicious HTTP GET requests while allowing other genuine HTTP GET requests generated from that system

***Index Terms* —** HTTP GET, Shortened URL, Kernel, Windows Filtering Platform

I. INTRODUCTION

Distributed Denial of Service (DDoS) attack uses a large number of computers to cause a coordinated DoS attack against a targeted web server. Flooding attack is usually caused by continuously sending multiple requests to bring down the targeted server. This eventually fills the server's buffer space. Once the buffer memory is full no further connections can be made to the server, thus making the service unavailable. This scenario is attained by misusing the URL shortening service which eventually leads to the formation of HTTP botnet [1]. URL shortening service is a technique, used in the World Wide Web (WWW) to substantially shorten a lengthy URL into a shorter one while redirecting the user to the requested webpage. This is achieved by referring HTTP redirect information from the available database of the URL shortener tool. This redirect information is required for redirecting the shortened URL to the requested original URL. Thus attacks could be launched using shortened URL. On making use of shortened URL service the user requested web page gets loaded in the web browser along with malicious scripts downloaded from the attacker. This is done by misusing the html ***iframe*** tag [2]. Thus HTTP GET flooding attack is initiated in the background by using the client side scripts downloaded from the attacker without interrupting the user's browser activity [3]. The attack will continue as long as the web page is active and it will not leave any trace of code unless the cache of the browser is not cleared on the client side.

The rest of the paper is organized as follows: Related work is discussed in Section II. Section III discusses the Threshold Based Kernel Level HTTP Filter (TBHF) algorithm in detail. HTTP GET attack generation and analysis are explained in Section IV. Performance analysis of TBHF is discussed in Section V and finally, Section VI concludes the paper.

II. RELATED WORK

Client side prevention of browser initiated flooding attacks is mainly done by Client side Caching optimization and implementation of the Human Interaction Proof (HIP) protocol [4]. In client side, a cache is maintained at browser level. All HTTP GET requests which are sent for the first time are cached in the local persistent storage [5]. Subsequent requests to the same

URL are served from the local cache. However, this method fails when the attacker requests different URLs, thereby making the technique ineffective.

Another method is Completely Automatic Public Turing tests to tell Computers and Humans Apart (CAPTCHA) [6] where the user is puzzled with an image, which contains randomly skewed alpha numeric characters. This is used to distinguish between human and bot behaviour. In the normal scenario when a host is making a large number of requests to a specific URL, the system detects an anomaly and redirects the user to a CAPTCHA page. If the user is able to complete the challenge, then the actual URL is served again. In the case of a bot requesting the URL, the CAPTCHA cannot be cracked, leading to inaccessibility of the URL. But in the current HTTP-GET attack scenario, the host requests different pages under the same server. Hence, no anomaly is raised and the CAPTCHA page is not served thereby surpassing the technique which leads to the flooding attack. Techniques such as blackholing are used by ISPs to mitigate DDoS attacks, where in traffic to a specific IP is blocked [7]. The blocked traffic may also contain genuine packets thus making the service unavailable to the legitimate user. Access Control Lists (ACLs) [8] are altered in order to stop users from participating in DDoS attack. But this technique becomes inefficient when attackers spoof genuine IP address.

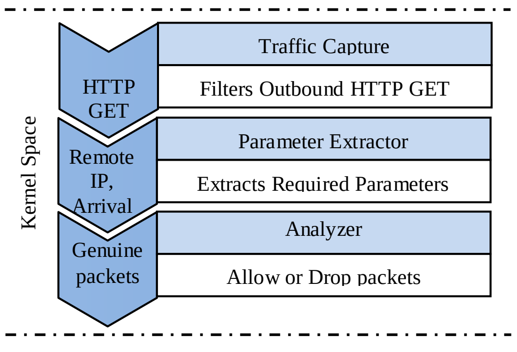

Figure.1 Block schematic of Threshold Based Kernel Level HTTP Filter

III. Proposed Work

A. Threshold Based Kernel Level HTTP Filter (TBHF)

The block diagram in Fig. 1, shows the different modules of the proposed TBHF, viz., Traffic capture, Parameter extractor, and the Analyzer module. First the packets are captured at kernel level by the traffic capture module. The output of the traffic capture module, the outbound TCP packets alone, are filtered and sent to the parameter extractor module which extracts the features such as remote IP address and the arrival time of packets. Then the packets are subjected to the Analyzer which decides whether to drop or allow the packets into the network based on the threshold set.

The analyzer module contains an IP frequency list to store the number of occurrences of individual IP address over a period of time. It checks the frequency of each IP address in IP frequency list and further decides whether to block or allow the packets.

The flow chart for the basic functioning of TBHF is given in Fig. 2. Threshold value as mentioned in section IV is set to restrict the number of HTTP requests to a particular IP address for a given period of time. Based on the parameters extracted from the packets ΔT values are calculated, which gives the time interval between current and previous instance of a packet for a particular IP address. Based on the threshold set if the value of the IP frequency list exceeds the threshold value, then the packets are dropped else they are allowed into the network.

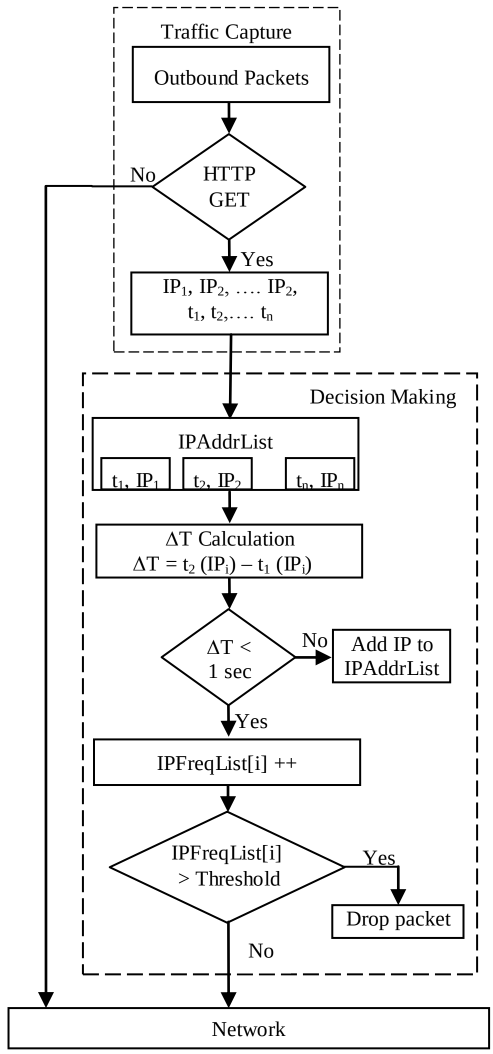

Figure.2 Functioning of TBHF

B. Proposed TBHF Algorithm

The proposed TBHF algorithm goes through the following steps: The network traffic (which includes all varieties of packets) is given as input to the TBHF. In Step 1 the outbound HTTP GET request packets are filtered and it is sent through the other modules for further processing. In Step 2, the parameters such as remote IP address and arrival time of packets are extracted from the filtered packets. The presence of the extracted IP address is checked in the IP address list and accordingly the IP frequency list is updated. If the IP address is not found in the IP address list, then the new IP address and its corresponding arrival time are stored in the IP address list. Then in Step 3, the difference in time (interval) between two packets of same IP address (ΔT) is calculated. If the time interval is less than 1 second, then the IP frequency list value for the corresponding IP address is incremented by 1. Otherwise the IP address is added to the IP frequency list. The time interval to refresh list is set to 1 second and also the IP frequency list values are reset to null after the elapse of every second. The threshold value (N) is set to 20 based on the experiments conducted in our lab, viz., only 20 HTTP GET requests are allowed to a particular IP in one second. The HTTP GET packets are allowed to an IP address till the corresponding IP frequency list value reaches N. If the IP frequency list value for an IP address exceeds N, then the packets are dropped. The value of N was set based on our research experiments conducted in various environments and their results are listed in Table 1.

Step 1:

Input: Network ***Traffic***

```
IF (Outbound packets)
    THEN
        IF (Packet == HTTP GET)
            THEN

        Step 2:
        //ExtractParameters
        // IP1, IP2, …, IPi - remote IP address
        // t1, t2, …, ti - Arrival time of packets
        //IPAddrList – IP Address List

                IPAddrList[IPi][0] = IPi;
                IPAddrList[IPi][1]=ti;

        Step 3:
        // ΔT - Difference in time between two instance of same IP address
        // N - Threshold value
        //IPFreqList – IP Frequency List

        ΔT = t2(IPAddrList[IPi][1]) – t1 (IPAddrList[IPi][1]);

        IF (ΔT < 1 second)
            THEN
                IPFreqList[i]++;
                IF (IPFreqList [i] < N)
                    THEN
                    Allow packet to Network;
                    ELSE
                        Drop packet;
                END IF
        END IF

    ELSE
        Allow packet to Network;
    END IF
END IF
```

IV. Work Done

A. HTTP GET attack generation

The Fig. 4 shows the log from Web Console tool in Mozilla Firefox on the Victim machine. Mark A shows the Link which is shortened using genuine URL shortener service provided by Google namely http://goo.gl. Mark B shows the Genuine URL, to which the user is redirected, to get the user requested web page.

Fig. 5 shows the log from Web Console using Mozilla Firefox during attack scenario. The Mark A in Fig. 5 shows the shortened URL that was provided by the attacker URL Shortening Service. In a genuine case when user clicks the link provided by shortened URL service, the user is redirected to the user requested web page but in case of an attack scenario, when a user clicks a malicious link provided by the attacker the user requested web page is displayed by exploiting the HTML iframe tag. Simultaneously GET requests are generated to victim server [9]. The Mark B shows the victim server IP address to which the HTTP GET Flooding attack has been initiated. The Mark C shows the dynamic resource name hich is intended to overcome the browser level caching mechanism. Dynamic resource names are randomly generated unique IDs to access the resources from the web server.

WAMP server was set up in attacker side to host the malicious web page. On clicking the malicious link, the script to generate HTTP GET attack was downloaded from attacker web server to the victim browser. The script did not install any malicious software in the victim machine rather used the browser script engine to generate the malicious requests.

TABLE 1 – HTTP GET ATTACK RESULTS *

Web browsers used	Total No. of Packets (for 30 second)	HTTP GET requests (for 30 second)	Average HTTP GET request (per second)
Safari	32184	15891	529
Google Chrome	59507	29708	990
Mozilla Firefox	87442	43654	1455

* Results shown for 30 second time interval during attack period

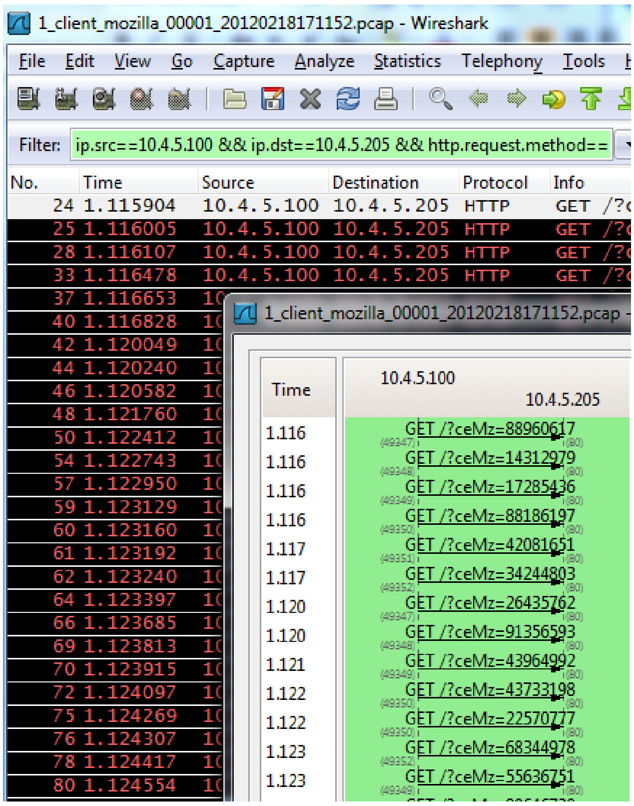

Figure.3 Graph analysis of HTTP GET attack using Wireshark

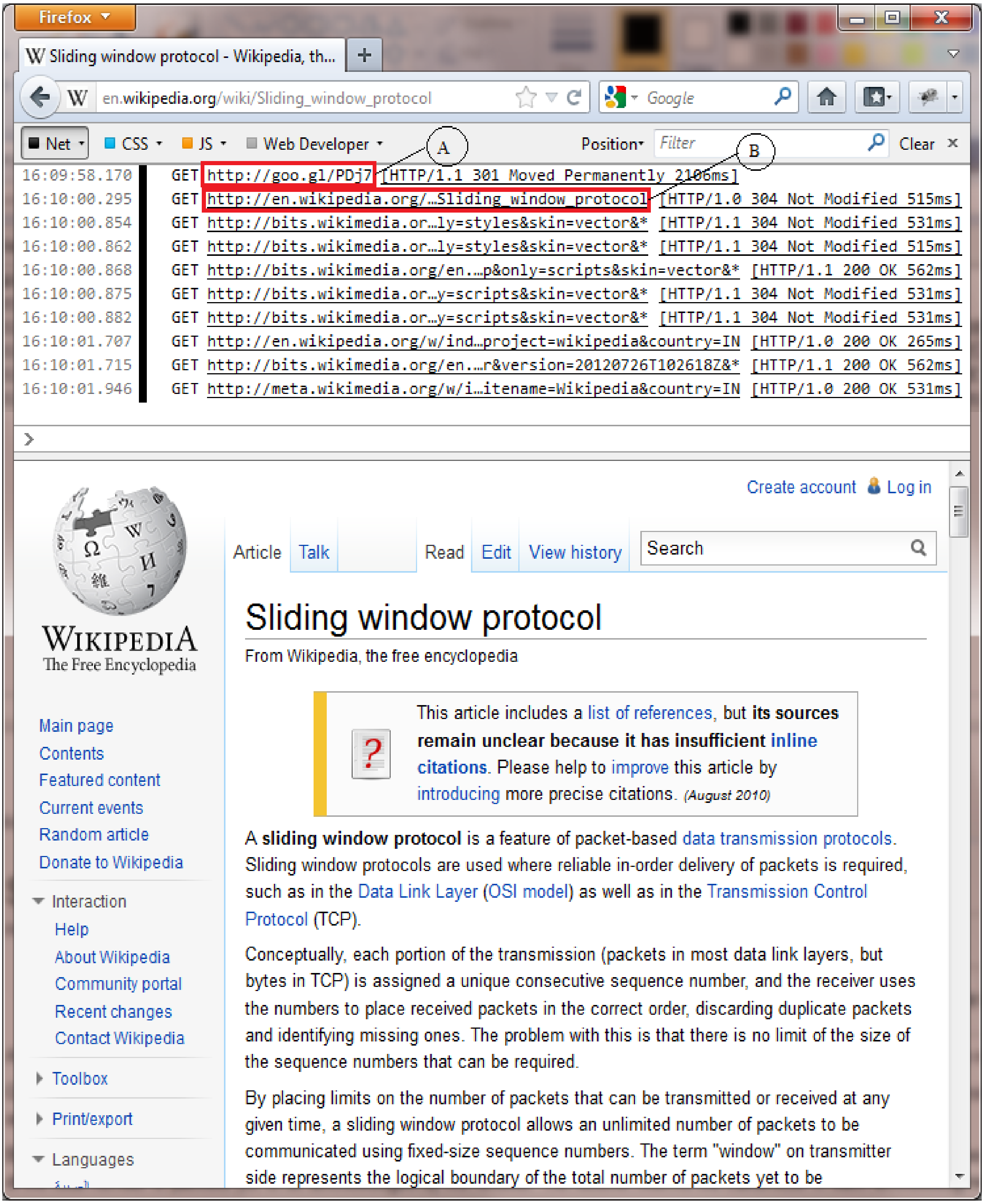

Figure.4 Web Console log using Mozilla Firefox during normal scenario

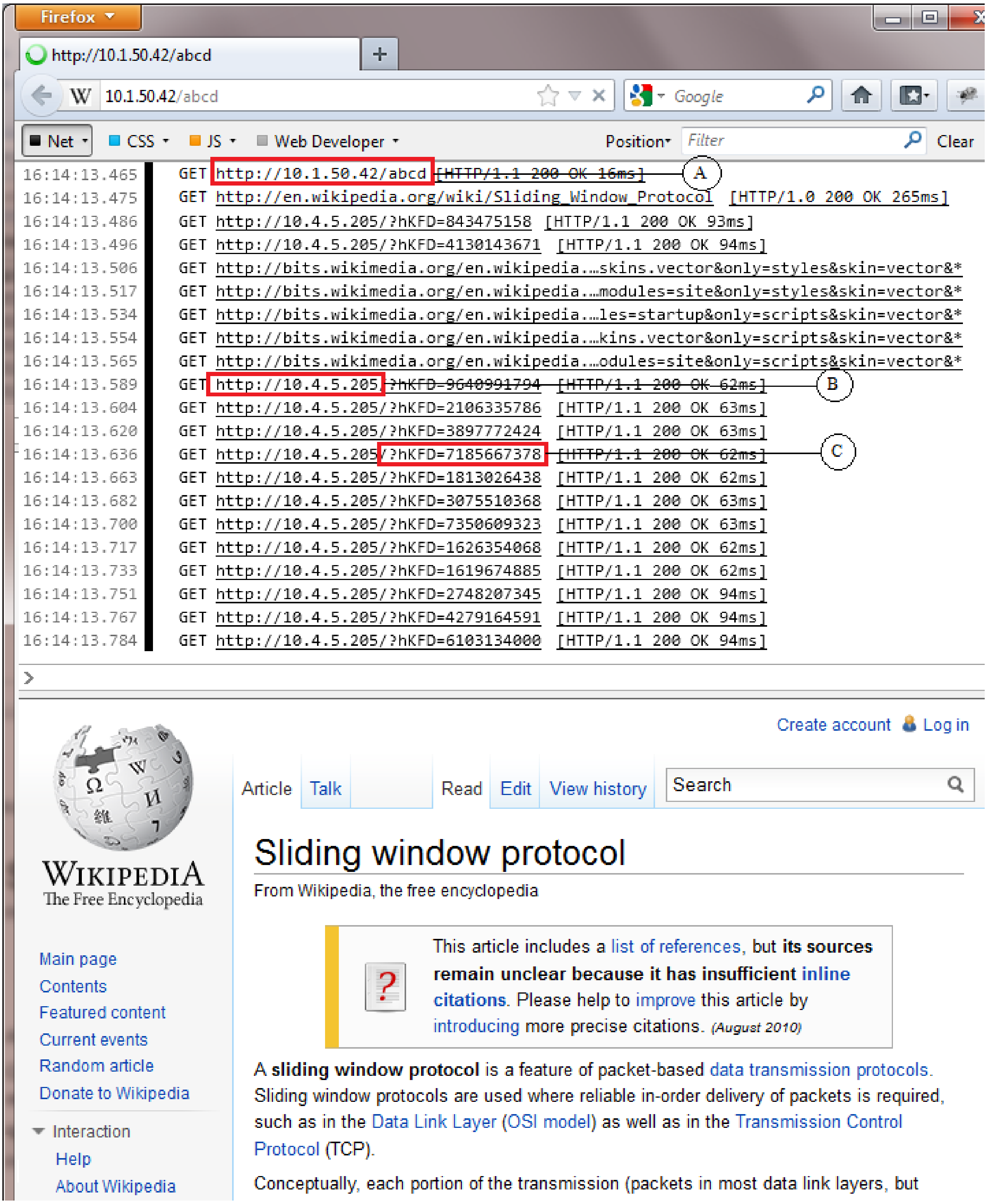

Figure.5 Web Console log using Mozilla Firefox during attack scenario

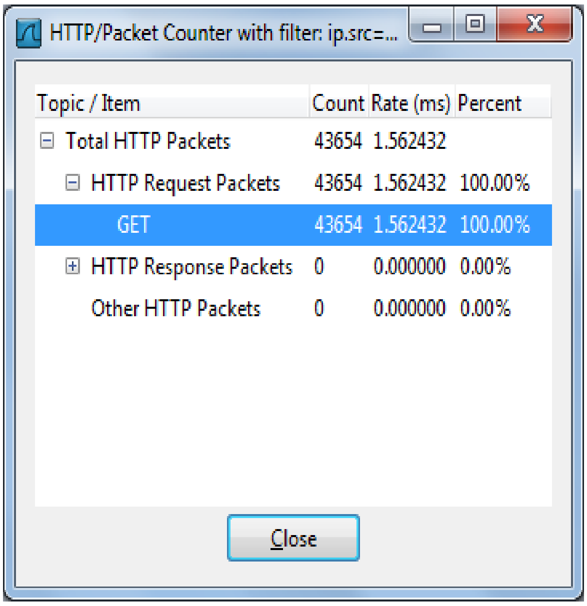

Figure.6 Statistics of HTTP GET attack before installing TBHF

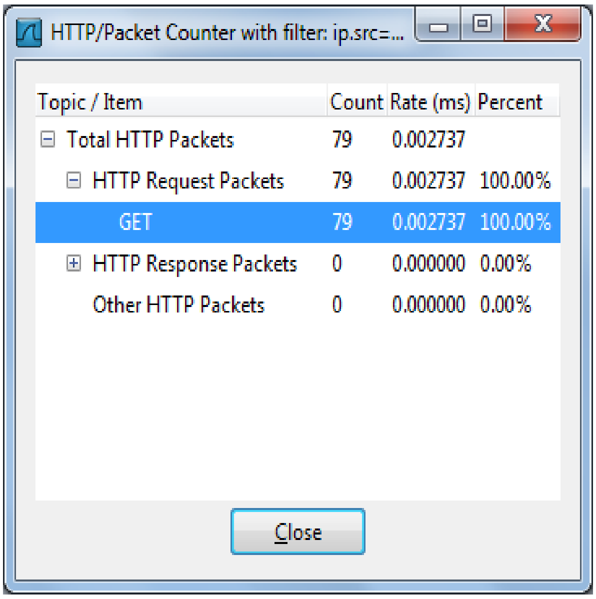

Figure.7 Statistics of HTTP GET attack after installing TBHF

TABLE 2 – PERFORMANCE ANALYSIS**

PACKET CAPTURE STATISTICS IN VICTIM MACHINE (30 SECONDS)					
Total No. of Packets		OUTBOUND TRAFFIC			
		Total No.	To target server	HTTP GET PACKETS Total No.	HTTP GET PACKETS To target server
Before	87442	43692	43685	43656	43654
After	371	132	112	83	79

** Results shown for 30 second time interval before and after loading TBHF filter

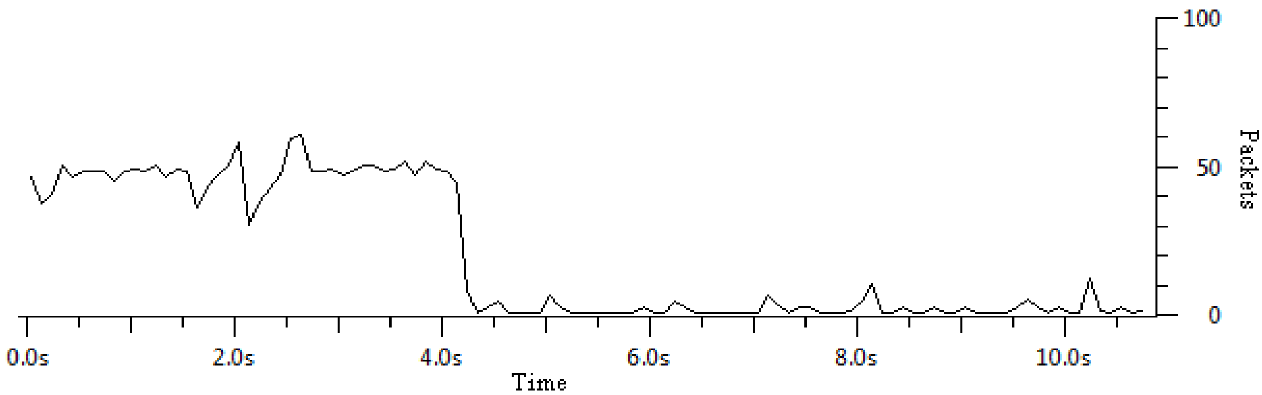

Figure.8 Suppression of HTTP GET attack traffic after TBHF installation

3436-5E'VBNM,alysis

When observed using Wireshark, an enormous amount of HTTP GET requests were generated to the target IP address in a very short span of time which usually does not happen in a genuine case. Based on this observation, a threshold was set to ascertain the number of HTTP GET requests generated to a particular IP address in a certain time period.

V. PERFORMANCE ANALYSIS

A. Experimental setup

The effective working of the proposed system was tested by generating HTTP GET attack in Local Area Network (LAN) and using the TBHF driver for its mitigation. HTTP GET attack was generated using client side scripting in victim web browser. Traffic analysis was done with respect to HTTP packets using Wireshark. Based on the obtained results, the parameters to be set for filtering out malicious HTTP GET packets using TBHF were decided. The same was implemented and the performance of the TBHF was studied.

B. Detection module at victim machine for kernel level packet filtering

Sun virtual box with Windows 7 installed in it was made the victim. The developed TBHF driver was installed on the victim machine. A GUI called Open System Resources (OSR), was used to load or unload the packet filter in the victim machine. The loading or unloading of the filter can also be done manually which involves making registry changes and then starting the filter through command prompt.

C. Results

A real time HTTP GET attack was generated in the victim machine. Table 1 shows the web traffic that was generated using three different browsers, viz., Safari, Google Chrome, and Mozilla Firefox.

Table 1 comprises of the total number of packets, number of HTTP GET packets sent to the targeted web server, and the average number of HTTP GET requests sent per second. The variation in the request generation depends on the capability of the browser script engine and system configuration. A Core 2 Quad processor with 4 GB of RAM running Windows 7 Operating System (OS) was made the victim machine.

Once the attack was initiated, traffic was captured using Wireshark and is shown in Fig. 3. The graph in Fig. 3 showing the attack packets, reveals the presence of continuous bombarding of HTTP GET packets from victim machine to the target server.

Fig. 6 shows the statistics of HTTP attack before installing the TBHF filter driver. In Fig. 6, the highlighted portion shows the packet count of HTTP GET packets sent from the victim machine to the target server before loading the TBHF filter driver for the interval of 30 seconds.

Fig. 7, shows drastic drop in the http packet count between the victim and target server over the same period of time after installing the TBHF filter driver. Thus from Fig. 7, it is evident that the developed filter was able to mitigate the DDoS attack.

The performance analysis of the proposed TBHF was done by measuring the number of outbound packets in the victim machine before and after loading the filter. Table 2 shows the number of HTTP GET requests sent to the target server before and after the filter installation. It is evident from the 2nd row that the flooding packets generated to target server were dropped by TBHF. The accuracy in terms of mitigating the DDoS attacks by TBHF works out to be 99.82% which is calculated as the ratio of the number of HTTP GET attack packets blocked to the total number of HTTP GET attack packets generated (43575 / 43654 * 100).

Fig. 8, shows the decline in the attack traffic after driver installation. X-axis shows the time in seconds and Y-axis shows the number of HTTP GET packets. It is observed that after the driver installation there is a drastic drop in flow of attack traffic which signifies that the HTTP GET attack is mitigated.

VI. CONCLUSION

Application layer attack has become a major threat to the internet in today's world. The focus of this paper was to come out with an effective solution for the detection and prevention of clients from inadvertently taking part in such attacks. Accordingly, a Threshold Based Kernel Level HTTP Filter (TBHF) was proposed and implemented in Windows 7 OS. Experiments were conducted by generating HTTP GET attacks and using TBHF for its mitigation. It was evident that the TBHF suppressed the flooding packets and thus prevented the client system from taking part in such an attack. The ongoing work is to implement TBHF in other OS and mobile platforms.

ACKNOWLEDGEMENT

The authors thank the National Technical Research Organization (NTRO), New Delhi, Government of India for sponsoring this research work under the Collaborative Directed Basic Research in Smart and Secure Environment project.

REFERENCES

[1] www.infosecisland.com/blogview/10442-DDoS-Attacks-Possible-via-URL-Shortener.html.

[2] Patsakis C, Asthenidis A, Chatzidimitriou A., "Social Networks as an Attack Platform: Facebook Case Study", Eighth International Conference on Networks, ICN '09, March 2009: p. 245-247.

[3] Takeshi Yatagai, Takamasa Isohara, and Iwao Sasase, "Detection of HTTP-GET flood Attack Based on

Analysis of Page Access Behavior”, IEEE PACRIM '07: p. 232–235.
[4] Daniel Lopresti, “Leveraging the CAPTCHA Problem”, Second International Workshop on Human Interactive Proofs, Bethlehem, PA, May 2005, Vol. 3517/2005 p. 97-110.
[5] www.developers.google.com/speed/docs/best-practices/ caching#LeverageBrowserCaching.
[6] Huy D. Truong, Christopher F. Turner, Cliff C. Zou, “iCAPTCHA: The Next Generation of CAPTCHA Designed to Defend Against 3rd Party Human Attacks” IEEE International Conference on Communications (ICC), June 2011, p. 1-6.
[7] J. Van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, J. Mulligan, H. Nguyen, M. Nguyen, A. Ramarajan, S. Saad, M. Satterlee, T. Spencer, D. Toll, S. Zelingher, “Dynamic Connectivity Management with an Intelligent Route Service Control Point”, INM '06 Proceedings of the 2006 SIGCOMM workshop on Internet network management, September 2006, p. 29-34.
[8] www.cisco.com/en/US/prod/collateral/vpndevc/ps5879/ps6264/ps5888/prod_white_paper0900aecd8011e927.html.
[9] www.secureworks.com/research/threats/botnet.

Mr Mohamed Ibrahim AK has completed his B.E. (Computer Sci. and Eng.) at Trichy Engineering College, Anna University, Chennai, Tamil Nadu. He has a keen interest towards coding and his field of interest includes Network Security.

Mr Lijo George has completed his B.E. (Computer Sci. and Eng.) at Trichy Engineering College, Anna University, Chennai, Tamil Nadu. His interest includes coding for cyber security.

Ms Kritika Govind received her B.E. (Computer Sci. and Eng.) from Sakthi Mariamman Engineering College, Anna University, Chennai, Tamil Nadu, in 2009. She is working as a Research Assistant at Department of Computer Science and Engineering, National Institute Technology (NIT), Tiruchirappalli, Tamil Nadu. She is also pursuing her Master of Science (MS by Research) in Computer Science and Engineering at NIT, Tiruchirappalli. Her interests include: Cyber security and network security.

S. Selvakumar is a Professor in the Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli, Tamil Nadu, India. He received his Ph.D. from the Indian Institute of Technology Madras (IITM), Chennai in 1999. His research interests include group communication in high-speed networks, routing, multimedia communication, scheduling for QoS guarantee, mobile networks, network security, wireless sensor networks, and network computing. He has to his credit of publishing 54 research papers. He is currently the Investigator of the Collaborative Directed Basic Research–Smart and Secure Environment (CDBR-SSE) Project sponsored by NTRO, Government of India, New Delhi. He is presently the member of All India Board of IT Education, AICTE, New Delhi.