I.J. Intelligent Systems and Applications, **2018, 4, 29-40**
Published Online April 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2018.04.04

Fuzzy-based User Behavior Characterization to Detect HTTP-GET Flood Attacks

Karanpreet Singh and Paramvir Singh
Dr B R Ambedkar National Institute of Technology, Jalandhar 144011, Punjab, India
E-mail: karanpreet.13.cse@nitj.ac.in, singhpv@nitj.ac.in

Krishan Kumar
University Institute of Engineering & Technology, Panjab University, Chandigarh, India
E-mail: k.salujauiet@gmail.com

Received: 31 March 2017; Accepted: 14 July 2017; Published: 08 April 2018

Abstract—Internet was designed to serve the basic requirement of data transfer between systems. The security perspectives were therefore overlooked due to which the Internet remains vulnerable to a variety of attacks. Among all the possible attacks, Distributed Denial of Service (DDoS) attack is one of the eminent threats that target the availability of the online services to the intended clients. Now-a-days, attackers target application layer of the network stack to orchestrate attacks having a high degree of sophistication. GET flood attacks have been very much prevalent in recent years primarily due to advancement of bots allowing impersonating legitimate client behavior. Differentiating between a human client and a bot is therefore necessary to mitigate an attack. This paper introduces a mitigation framework based on Fuzzy Control System that takes as input two novel detection parameters. These detection parameters make use of clients' behavioral characteristic to measure their respective legitimacy. We design an experimental setup that incorporates two widely used benchmark web logs (Clarknet and WorldCup) to build legitimate and attack datasets. Further, we use these datasets to assess the performance of the proposed through well-known evaluation metrics. The results obtained during this work point towards the efficiency of our proposed system to mitigate a wide range of GET flood attack types.

Index Terms—GET flooding, application layer, anomaly detection, denial of service.

I. Introduction

Defending against Denial of Service (DoS) attacks have always been an arduous challenge for security firms due to the stateless nature of the Internet [1]. DoS attack targets the availability of the online services by flooding a large number of unsolicited packets to overload victim's network and transport layer resources. DoS attacks not only targeted the working of traditional network stack but also have threatened modern day cloud networks [2]. As more and more solutions are being offered to confront denial of service attacks, these attacks are continuously adopting sophisticated methodologies to circumvent such solutions. The intention of an attacker in DoS attack is to forbid service access to legitimate clients in comparison to some traditional network attacks, which primarily aim at stealing and misusing the confidential data. During an attack, the victims' services become unavailable until the attack terminates or is effectively mitigated. The attack might last for a few seconds or even days which lead to huge financial losses to the service providers. The motives behind such types of attacks are generally associated with financial gains, business competitions, political gains, etc.

To exacerbate the situation, attackers came up with Distributed Denial of Service (DDoS), which refers to initiating an attack using a large number of bots (compromised systems) simultaneously in contrast to a single attack source in basic DoS attack, as illustrated in Fig. 1. The attacker creates a network of compromised systems over the Internet known as botnet [3]. This network is controlled by the attacker hidden behind several layers of bots known as stepping stones with the intention to challenge its identification. The attacker initiates the attack process by disseminating commands to the compromised systems. These systems then further launch the actual attack flow towards the victim in accordance with the received commands. The damage caused by such an attack is substantially greater than that of a simple DoS attack as the amassing of traffic from multiple sources guarantees a high amplitude attack.

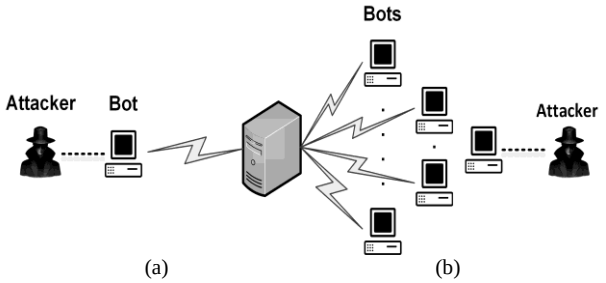


Fig.1. (a) DoS attack, (b) Distributed DoS attack.

Copyright © 2018 MECS

Typically, a DDoS attack falls into one of the following categories: volume-based attacks, protocol attacks or application layer attacks. Volume based attacks, commonly known as flooding attacks direct a large amount of unsolicited traffic towards the victim resulting in exhausting either the network or the processing capacity of the victim. These include attacks such as ICMP flood, UDP flood, etc. Attacks that exploit the vulnerabilities of protocols functioning at different layers of TCP/IP stack are classified as protocol attacks. These include attacks such as ping of death, teardrop, etc.

Finally, application layer attacks misuse configurations and functionalities operating behind various applications running on the Internet. Some common application layer attacks are Slowloris, HTTP GET flood, etc. DDoS attacks now-a-days are predominantly launched by organized groups of hackers. Furthermore, there had been cases where a decentralized group of hacktivists launched a series of DDoS attacks to protest against any action taken by the government or companies. Few major attacks of the year 2016 is shown in Fig. 2 [4].

A. Motivation

Present-day attacks, instead of exploiting network and transport layer vulnerabilities, have risen to the higher layer of TCP/IP i.e., application-layer. In recent years, attackers are now continuously exploring application layer DDoS attacks as their detection requires security firms to put on significant research efforts. HTTP-GET flood DDoS attack is one of the most common form of application layer abuses, where the bots attempt to mimic browsing behavior of a legitimate client but with amplified request rates [5]. The attacks launched on network or transport layer are different from that of application-layer attacks in the following ways:

- Network layer attack detection techniques are not able to extract enough information from packets so as to report an application layer DDoS attack.
- A successful TCP connection is required to launch an application layer DDoS attack, thus, evading the transport level detection mechanisms.
- Attackers use legitimate IP address, therefore the anomaly detection schemes based on spoofed IP addresses will also not work [6].
- Mimicking the client access behavior defies the logic behind many of the existing attack detection schemes.

A large part of online services provided by the organizations are dependent on the working of HTTP for interacting with their potential clients. Consequently, “application layer DDoS attacks” generally refers to HTTP-GET flood attacks in the present literature [7]-[10]. It may be noted that these two terms are used interchangeably in this text. Our work aims at efficiently detecting the bots that are responsible behind HTTP-GET flood attacks.

B. Contribution

Initially, we carefully design a two detection parameters that take into consideration behavioral aspects of clients to quantify their browsing history. These parameters depict the legitimacy of a client based on its behavior when browsing a website. We propose a complete mitigation framework that uses these parameters to detect GET flood attacks. The detection takes place at the client level i.e., every client is individually monitored for possible anomalies. Due to the absence of application layer DDoS attack traffic, we make use of open source software tools and benchmark datasets to fabricate the attack traffic on a specially designed experimental setup.

The paper is structured as follows. Section II discusses the related works on the detection of application layer DDoS attacks. Section III introduces the proposed detection parameters and mitigation framework. Section IV discusses various phases of the experimental setup and provides a brief summary on preliminary analysis of benchmark datasets. Section V follows with a thorough analysis and discussion of the results obtained from our experiments. Finally, Section VI concludes this work along with some comments on the possible future work.

Fig.2. Major DDoS attack incidents in the year 2016.

II. Related Work

Based on our previous study [11], the literature on detection of application layer DDoS attacks can be classified according to their respective underlying detection methodologies i.e, queue management, popularity, challenge, score, etc. This section outlines the work done in the area of application-layer DDoS attack detection and its mitigation in chronological order.

Copyright © 2018 MECS

Jung *et al.* [12] offered enhancement to Content Distribution Networks (CDNs) in order to distinguish a DDoS attack from a flash crowd. The authors identified two key properties associated with a flash event – there is an increase in number of clients in flash event as compared to DDoS attack where the traffic is generated from a small set of IP clusters; largely the old set of IP clusters are responsible for traffic generation in case of flash event whereas new IP clusters are formed during a DDoS attack. Hopper *et al.* [13] and Kandula *et al.* [14] proposed graphical puzzle based bots detection schemes in 2003 and 2005 respectively. The clients are asked to solve CAPTCHA before allowing any resource access.

Yen *et al.* [15] proposed statistical based approach in which the server maintains the client's recent request history. The proposed system is divided into three phases. Initially, a client is considered as suspicious based on the frequency of repetitions in its request pattern. Then the requested objects are identified based on which the attackers are distinguished from legitimate clients. Ranjan *et al.* [16] proposed a counter mechanism based on deviation of a client session characteristics from the legitimate behavior. A suspicion value is assigned to each client session proportional to the deviation in terms of session arrival, request arrival and workload characteristics. The scheduler then decides when are where to serve sessions based on their suspicion measures.

Yu *et al.* [17] proposed a mechanism that integrates detection and encouragement scheme into a Defense and Offense Wall (DOW) model. The detection system is based on K-means clustering algorithm to detect anomalous connections that are dropped by the server after characterization. The encouragement system requests the clients to increase their session rate which increases the probability of their requests being served by the web server. Srivatsa *et al.* [8] integrated admission and congestion control mechanisms to defend against application-layer DDoS attack. They used JavaScript on the client's browser to embed a 16 bit value known as authenticator in the port number field of TCP header. Based on this value, the attack packets are filtered at the network layer of the victim.

Mirkovic *et al.* [18] proposed a method to characterize clients' legitimate behavior based on request dynamics like request inter-arrival time, etc. and content access priority like request sequence, etc. The deviation of the current client session from predefined legitimate behavior characterizes it as an attack. Wen *et al.* [19] proposed an architectural extension to distinguish surge from recursive and repeated application-layer DDoS attacks based on entropy of incoming source and target webpages. The system initially detects for an anomaly against normal behavior modeled using static autoregressive model and Kalman filter.

Das *et al.* [20] identified different application-layer DDoS attacks using three different detection modules. The value of HTTP request arrival rate calculated in a HTTP window signals one of the given scenarios-random flooding, shrew flooding and flash crowd. In 2011, Ankali *et al.* [21] proposed two attack detection mechanisms for HTTP and FTP based on HsMM. They extracted various parameters like request rate, page viewing time and requested sequence to model legitimate behaviour. Ibrahim *et al.* [22] designed a threshold based kernel level filter to detect URL based HTTP flood attacks. The filtering takes place based on the number of incoming requests from a particular client. Limkar *et al.* [23] detected application layer DDoS attacks using Hidden Markov Model (HMM). HMM is trained using the legitimate sequence of requests that a human client usually follows during browsing a website.

Sivabalan *et al.* [24] proposed a detection system in which the server load level is divided into three parts using two threshold values- low load threshold and high load threshold. CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) and AYAHs (Are You A Human) are occasionally generated during a session to create client signatures before and during a session. Wang *et al.* [25] extended their previous work [26] to support the modeling of legitimate behavior even from noisy datasets i.e., web traces mixed with traffic from web bots. The authors used density based clustering to identify web crawler traces in the training dataset. Giralte *et al.* [27] represented the legitimate client behavior in terms of layer 4 and layer 7 parameters like number of GET requests, GETs mean, mean of flows per client, standard deviation of flows per client, etc. A three stage model was designed to detect a variety of application-layer DDoS attacks wherein each stage was able to capture some of the attacks. Xie *et al.* [28] proposed a scheme that primarily detects web proxy based DDoS attacks using Hidden semi Markov Model. The authors captured temporal and spatial localities to model web proxies' access behavior using the server logs.

The popularity of a large website varies with time as the contents are regularly updated and deleted. In 2014, Wang *et al.* [29] proposed a dynamic popularity based DDoS detection scheme based on their previous work [26]. Large deviation principle characterizes the difference in expected and actual popularity of webpages. Zhou *et al.* [7] extended their previous work [19] to sustain under heavy backbone traffic conditions. To implement live detection, they used a real time frequency vector based on target's resource requests. Attack detection module is only triggered in case of an anomaly detected by the front end sensor which reduced the probability of frequent computations.

Liao *et al.* [30] proposed machine learning-based detection technique that used a support vector machine to identify presence of any attacks. The rhythm-matching algorithm is applied to identify similar patterns. Xiao *et al.* [31] used K-nearest neighbors algorithm to identify flows that may have occurred from same software or bots. Kshirsagar *et al.* [32] provided ontology of HTTP requested that could provide a means to detect GET flood attacks. However, sophisticated bots with the capability of producing request similar to a legitimate client can easily evade such ontology based detection systems. Kobayashi *et al.* [33] introduced a concept of fooling

Copyright © 2018 MECS

monitored attacks by installing normal and decoy servers to respond to legitimate client and bot requests respectively. This mechanism deceives an attacker by providing false information about the server's status.

Miu *et al*. [34] employed browsing behavior of users to capture anomalous sources. The authors used the access sequence of web pages by the users. The difference in actual and expected transition probability among web pages of every user is quantified. For this, the log likelihood is computer for every session as a metric for differentiating attack bots and legitimate users. The advanced and sophisticated bots can evade their detection scheme.

Web services are constantly under the threat of various forms of application-layer DDoS attacks which can anytime disrupt its normal functioning. An effective real-time defense against these rising threats is need of the hour that can ensure its continuous availability. It is important to detect and respond to an on-going attack in least possible time and keeping computational complexities under tolerable limits.

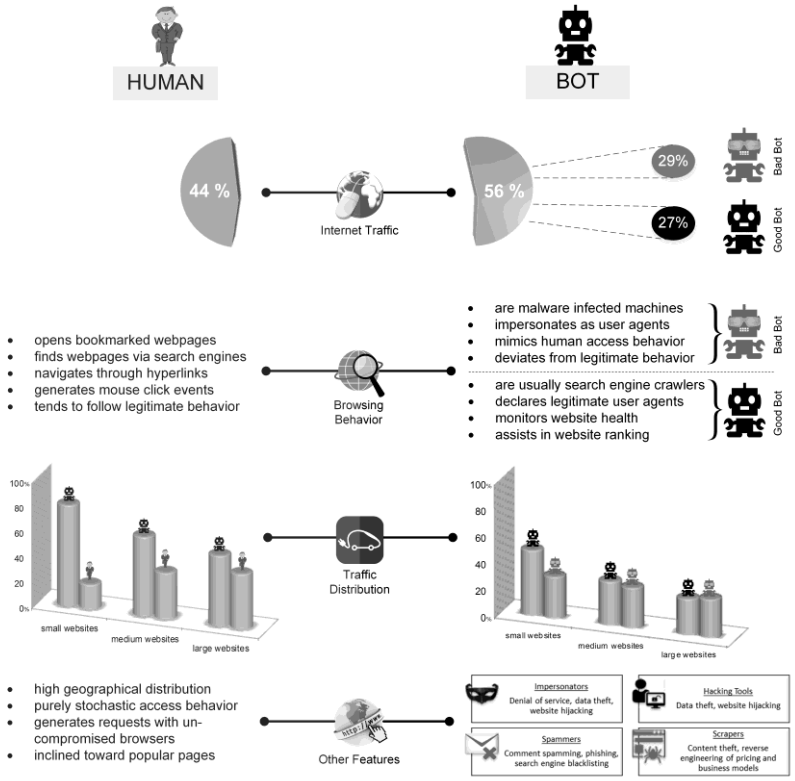


Fig.3. Characteristics of humans and bots.

We represent some if the characteristics possessed by humans and bots in Fig. 3. The primary objective of the detection system is to identify clients that do not adhere to normal browsing characteristics. Therefore, researchers target exploring unique behavioral features to allow discrimination among human and bot clients.

III. Proposed Detection Framework

Our proposed work aims at efficiently detecting and mitigating GET flood DDoS attacks against HTTP with minimum collateral damage. The proposed framework is represented in Fig. 4. Our work detects bots behind the attack using two parameters, Request Index (*IReq*) and Repetition Index (*IRep*), which operates on legitimate browsing semantics to differentiate among legitimate and bot clients. The values of these two detection parameters *IReq* and *IRep* are used as input to our fuzzy control system (FCS). Output from FCS is given to the request scheduler of a web server. The scheduler uses the results from FCS to decide whether to filter or process request of a particular client. The scheduler also has the capability of permanently blocking a client so as to reduce the load incurred while receiving its requests.

A. Detection Parameters

After a comprehensive analysis of browsing behavior, we propose two detection parameters to differentiate among bots and legitimate clients. For this work, we consider using two time windows small and large, with 30 seconds and 120 seconds durations respectively, for computation of behavior parameters. A single long window comprises four short windows. The detection parameters and their definitions are discussed below.

1) Request Index (IReq)

Typical client browsing semantics can be divided into two phases. In phase I, known as *uptime*, a client is requesting for web pages from the server. There are no requests created by the client during the second phase, also known as *downtime* or *viewing time*. In second phase,

a client spends time viewing all the requested web pages. The duration of these two phases varies differently for different clients. In first case, a client will send a large number of requests in one window following which it will send less or no requests in the next. Contrariwise, in other case, if a client sends fewer requests in one window, then it has a high probability of sending large amount of requests in the next window.

Request frequency (η) of each client in a short time window is calculated to obtain the cumulative percentage frequency distribution. Using the three sigma principle [35], this distribution is divided into four classes namely low, normal, high, and anomalous. These classes form a set known as *Request_Set (R)*. The request frequency of every client during a short time window is then mapped to the corresponding class from this set. Consider an example where a client makes 5, 15, 25, and 35 requests in four short consecutive windows. Additionally, assume that the class limits are computed as follows:

If (η > 0 && η <= 10) Then *Low*,
If (η > 10 && η <= 20) Then *Normal*,
If (η > 20 && η <=30) Then *High*, and

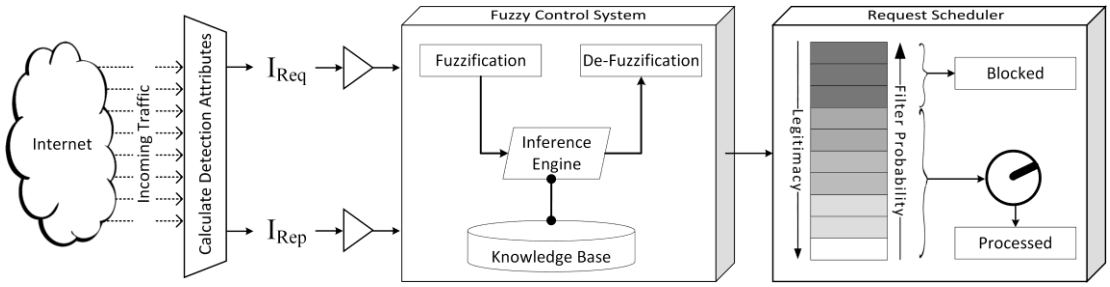


Fig.4. Characterizing incoming clients based on the proposed parameters.

If (η > 30) then *Anomalous*.

This makes a sequence of the client as Low → Normal → High → Anomalous. This sequence consists of three class shifts low to normal, normal to high, and high to anomalous. We calculate the frequency of the occurrences of all such class shifts. Each of the 12 possible class shifts is then assigned a score using Eq. (1) and Eq. (2).

$$\lambda = \frac{\ln(0.1)}{max(\alpha)} \tag{1}$$

$$\Theta(x, y) = 1 - e^{-\lambda \times \alpha(x,y)} \tag{2}$$

λ is known as the relaxing factor, α(x,y) defines the frequency of class shifts from class *x* to *y*. $\Theta(x, y)$ defines the score of class shift from class *x* to class *y*. The value of $\Theta(x, y)$ is normalized between a range of 0 and 11 to increase the scale of the values for easy interpretation. The value of Request Index (*IReq*) is averaged sum of four consecutive short windows.

$$I_{Req} = \sum_{i=1}^{i=3} \frac{\Theta(\varphi(\eta(T_i)), \varphi(\eta(T_{i+1})))}{l-1} \tag{3}$$

φ (n) maps request frequency n to the corresponding class and η(t) defines request frequency of a client in short window.

2) Repetition Index (IRep)

Usually a legitimate client will rarely request the web page that it has already visited a moment ago. Due to high request rates of bots during an attack, the likelihood of receiving repeated request for the same web pages by the server highly increases. It may be noted that the applicability of this scenario is limited to four short windows i.e., a single long window. We define a detection parameter Repetition Index (*IRep*) that actually computes the extent of repeated requests made by the clients. *IRep* is computed using the Eq. (10).

$$I_{Rep} = \frac{\sum_{\forall i \in \delta(T_n)} W(i)}{\sum_{i=1}^{i=4} \eta(T_n)} \tag{4}$$

$\delta(T_n)$ is the set of web pages requested in short window. W(*E*) is 1 if web page *E* has already been requested in a long window. Again, the value of *IRep* is normalized between a range of 0 and 11.

B. Fuzzy Control System (FCS)

A fuzzy set is represented as (X, μ) where *X* is a set and μ: $X \rightarrow [0, 1]$ is a membership function [36], [37]. For every $x \in X$, the value $\mu(x)$ is known as the grade of membership of *x*. We usually denote fuzzy set with membership values as

$$\{\mu(x_1)/x_1, \mu(x_2)/x_2, ..., \mu(x_n)/x_n\}$$

Membership function μ maps each element in *X* to a membership value between 0 and 1 [38]. There are three operational modules that constitute FCS. These modules are discussed below.

Copyright © 2018 MECS

1) Fuzzification

In this work, the fuzzy control system takes two crisp inputs *IReq* and *IRep*. For every client connected to the server, these values are fed to FCS after 120 seconds for further classification. *IReq* and *IRep* can take any values from ranging from 0 to 11. We use the membership function given in Eq. (5).

$$\mu_A(x) = \begin{cases} 1, x \le \alpha \\ \dfrac{\beta - x}{\beta - \alpha}, \alpha < x < \beta \\ 0, x \ge \beta \end{cases} \quad (5)$$

The values of α and β are different for *IReq* and *IRep*. These values are estimated from the process of threshold calibration through F-measure. Both the benchmark datasets WorldCup and Clarknet, have different values for α and β that is used to estimate the grade of each client.

2) Inference Engine

The inference engine computes the output set for each applied input based on a specific rule set. These rules are usually designed using the knowledge base. We define four rules shown in Table 1. Every entry in the table associates an output corresponding to the inputs shown in rows and columns (*IReq* and *IRep*).

Table 1. Inference rules

Rule Base		*IRep*	
		Low	High
IReq	Low	*Bot*	*Bot*
	High	*Bot*	*Human*

3) Defuzzification

The outputs from the inference engine are converted to crisp values during defuzzification. There are number of ways to perform defuzzification of a fuzzy quantity. We have chosen center of gravity method, where centroid of each membership function is initially computed. Following this, the final crisp value is computed using Eq. (6), which takes the weighted average of individual centroids. We term this value as *Legitimacy* of every client, which is fed to the scheduler that makes blocking and processing decisions.

$$Legitimacy = \frac{\sum_{x=\alpha}^{\beta} \mu_A(x) * X}{\sum_{x=\alpha}^{\beta} \mu_A(x)} \quad (6)$$

C. Scheduler

The scheduler is responsible to process requests received from the clients. It processes a large number of requests belonging to different clients within a single time window. During an attack, a server receives a large number of requests from bots. As a result, legitimate requests are either discarded or eventually get less computing time as compared to the latter.

We design a scheduler that uses the *Legitimacy* values of clients from FCS to filter out badly behaving clients. When under heavy load, the scheduler initially sorts the *Legitimacy* values of clients in increasing order. Following this, a set of clients that have very low *Legitimacy* value are blocked by the scheduler from further accessing the server. The proposed framework filters out anomalous users after every 120 seconds during an attack. This means that the attack mitigation is started only after 120 seconds before which the server continuously receives attack traffic.

IV. Experimental Setup

This section provides overview of various phases of the experimental analysis conducted under this work. Initially, we discuss the steps taken for dataset pre-processing in order to remove any irrelevant entries. Following this, the experimental design to prepare attack traffic traces is discussed. Finally, the attacks that have been studied in this work are explored.

A. Dataset Pre-Processing

In our earlier study on application layer DDoS attacks [11], it was observed that many works have used benchmark dataset such as WorldCup and Clarknet to model legitimate behavior. We also considered using these benchmark datasets into our work. The details of portions of benchmark datasets taken into consideration are given below.

- *WorldCup*: This dataset contains traffic for a period of 92 days (April 30, 1998 to July 26, 1998). We extracted two hours portion contain 956898 records with 10509 unique clients from the 42nd day to model legitimate behavior.
- *Clarknet*: This dataset spans over seven days (August 28, 1995 to September 3, 1995). We extracted 251334 records with 22569 unique clients from the complete dataset.

A web server maintains logs of each access made to its resources. The entries in the logs are represented as

1234 - - [17/Jun/1998:06:11:33 +0000] "GET /images/hm_score_border_r01.gif HTTP/1.0" 200 929

This record can be interpreted as follows. This server received GET request from client '1234' for an access to 'score.html' on 17th June 1998 at 06:11 am.

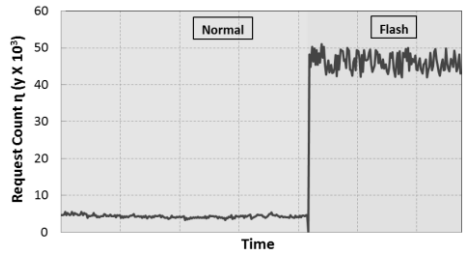
The code value 200 signifies successful request completion. We filter out the records with status code other than 200. Other records with missing information are eliminated to avoid influence on dataset analysis. Clients active for less than 120 seconds are also filtered out. Fig. 4 depicts the overall number of requests made

Copyright © 2018 MECS

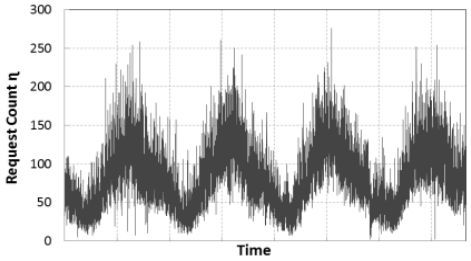
by clients for different datasets.

B. Design

Researchers often synthesize attack traces by making use of various software tools. After that, these traces are mixed with benchmark datasets in order to assess detection performance of various detection techniques. Based on our previous study [11], we designed an experimental setup to fabricate attack traffic traces, as shown in Fig. 5. This experimental setup consists of three machines with Intel Core i7-4790 processor running at 3.60 GHz equipped with 4GB RAM. First machine uses Apache JMeter[1] to generate attack and legitimate traffic. Second machine acts as an intermediate router and creates random delays from the requests received from the first machine before forwarding. Third machine uses Apache Server[2] to logs the accesses made by the clients emulated on the first machine.



(a) WorldCup



(b) Clarknet

Fig.4. Number of requests in short time window.

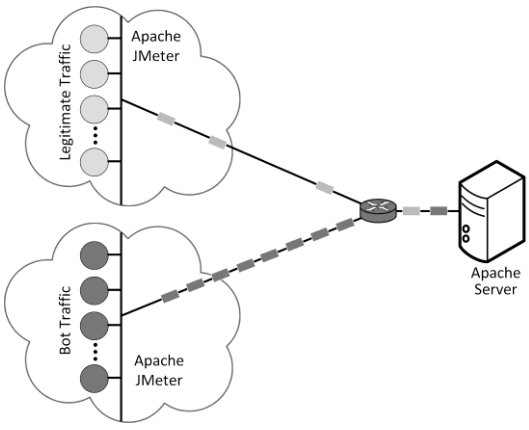


Fig.5. Setup for fabricating attack datasets.

During the pre-processing of benchmark datasets, we extracted the web object names and their respective sizes on the server. Using this information, the web pages and other web objects were replicated into the database of third machine using self-written script. This way, the trace information collected from the accesses made by legitimate clients and bots from the first machine is likely to portray a real-world scenario. We took 150 bots, which were assigned a pool of unique IP address, to generate the bot traffic [39], [40]. This set of IP address is further used to differentiate between legitimate and bot requests in the access logs during processing. GET flood attack is launched at inter-request delay of 100 ms, 200 ms, and 300 ms, chosen based on three renowned virus programs Netsky.Q, BlueCode.Worm, and Trojan_Sientok [11].

C. Attack Strategies

We fabricate a total of 8 different attacks using different configurations set in Apache JMeter. The details of these attacks are represented below. Three high rate attacks are generated (<H_100>, <H_200>, and <H_300>) with varying inter-request delays. The delays of these attacks were set to 100ms, 200ms, and 300ms respectively. Apart from this, we also produce three sophisticated attack types (<SP1_300>, <SP2_300>, and <SP3_300>) for a comprehensive evaluation of the proposed system. In these attacks, bots request only for popular web pages (except <SP3_300>) i.e. the web pages that are frequently accessed by the legitimate clients. We assume that this set of popular web pages is known to an attacker, which makes it possible to launch complex attacks. In <SP1_300>, bots make requests in no restricted sequence. In <SP2_300>, bots randomly request popular web pages having very small size. In <SP3_300>, bots request small sized web pages whether or not they are popular or unpopular. We also produce two asymmetric attacks (<AS_2500> and <AM_2500>) that generate requests for heavy sized web pages but at a low rate. Consequently, it becomes hard to detect such attacks when the detection system solely relies on the request rates of clients. In <AS_2500>, only single page is requested in a short time window, whereas in <AM_2500>, multiple heavy sized web pages are requested in short window with inter-request delay of 2500ms.

V. Results and Discussion

Table 2 represents the pseudo code that is used to compute the detection parameter values for different clients. In this section, we initially present the process of optimal threshold calibration and then discuss the results obtained from the experimental analysis.

A. Threshold Optimization

In order to compute the values of optimal thresholds, we used F-measure. We initially compute the values of the confusion matrix comprising four basic attributes true negative (TN), false positive (FP), false negative (FN), and true positive (TP). TNs are legitimate clients in

[1] http://jmeter.apache.org/

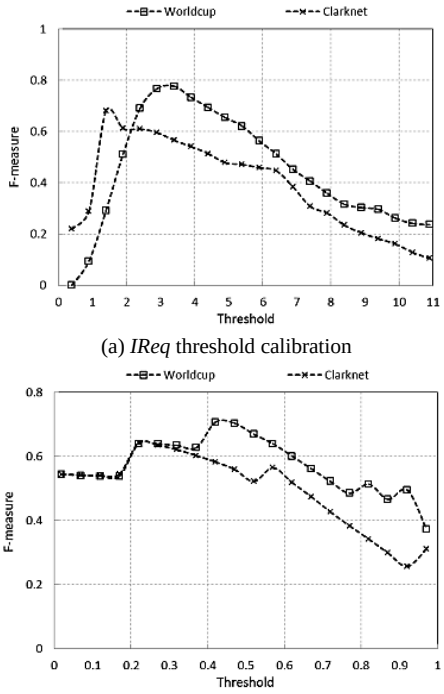
[2] https://httpd.apache.org/

Copyright © 2018 MECS

legitimate zone, FPs are the legitimate clients in attack zone, FNs are the bots in legitimate zone, and TPs are the bots in attack zone. Fig. 6 represents the plot of F-measure against different threshold values.

Table 2. Pseudo code for computation of detection parameters

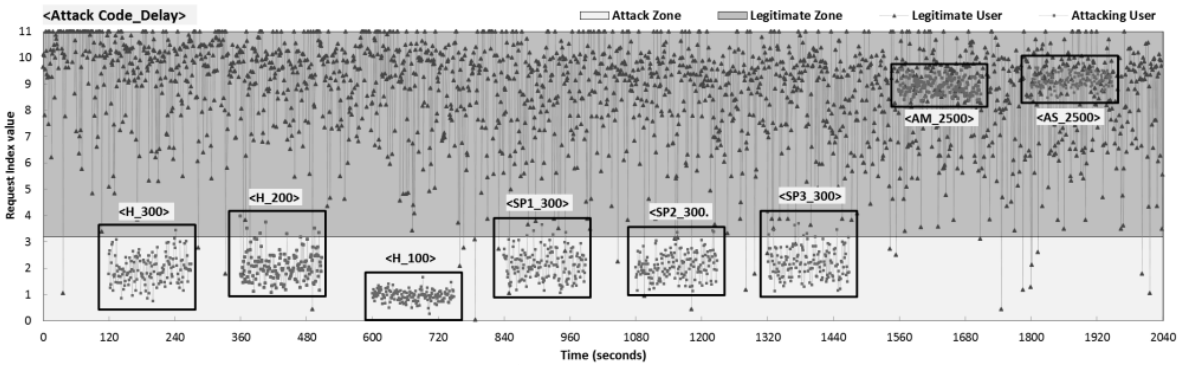
Pseudo code to compute detection parameter values of clients
01: Initialize short and long time windows as 30 and 120 seconds
02: **FOR EACH** client session
03: Split each session into sub-session of 120 seconds
04: **FOR EACH** client sub-session
05: Split sub-session further into four part of 30 seconds each
06: **FOR EACH** sub-session of length 30 second
07: Map the request frequency to its respective class
08: Cumulatively add *IReq* using Eq. (3)
09: **END**
10: Average *IReq* by dividing it with four
11: Calculate *IRep* using Eq. (4)
12: **END**
13: **END**



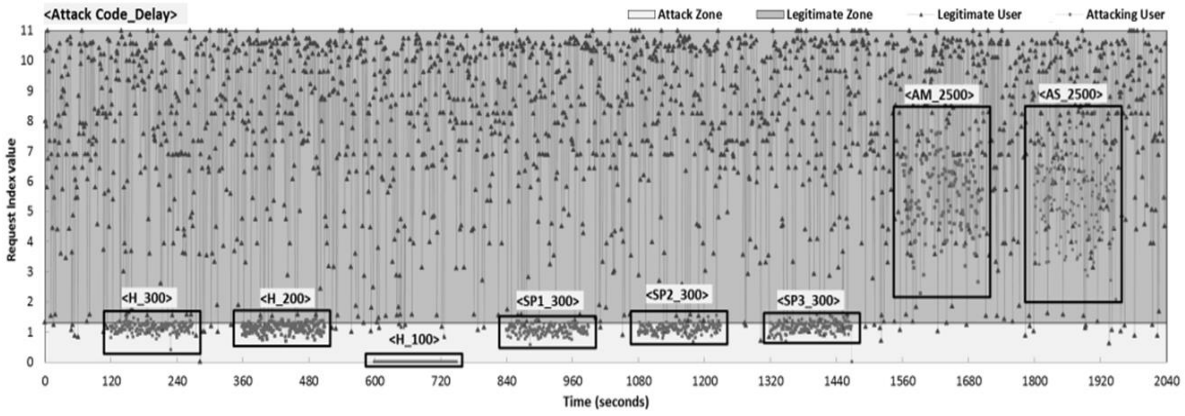
(a) *IReq* threshold calibration

(b) *IRep* threshold calibration

Fig.6. Threshold calibration using F-measure.

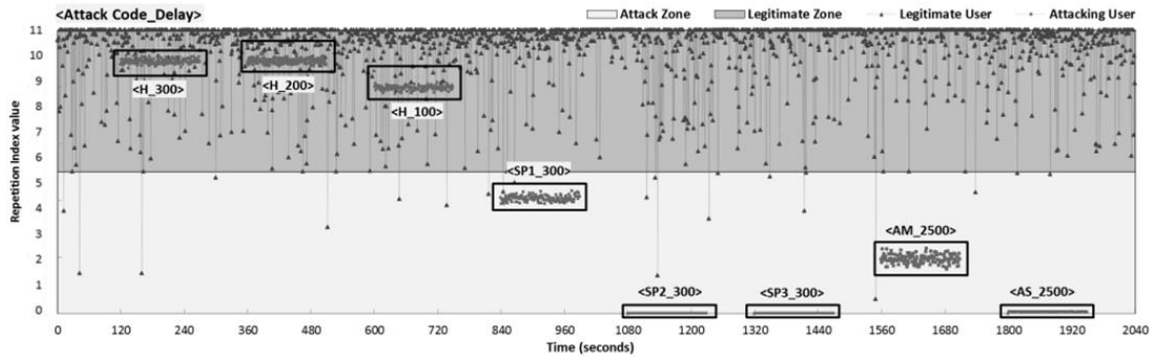


(a) WorldCup

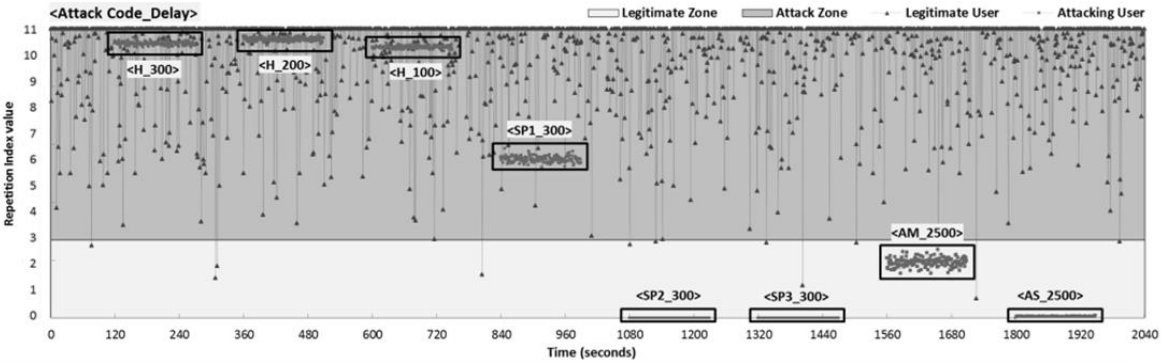


(b) Clarknet

Fig.7. IReq values of legitimate clients and bots.

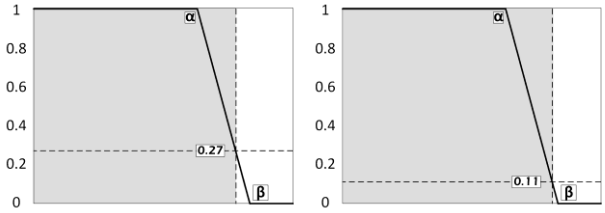


(a) WorldCup

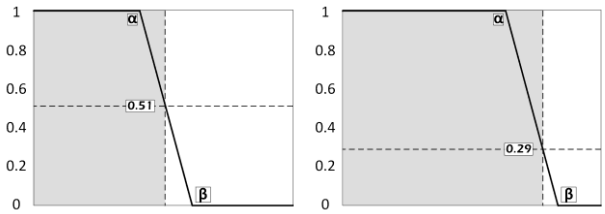


(b) Clarknet

Fig.8. *IRep* values of legitimate clients and bots.



(a) MF for IReq in WorldCup (b) MF for IRep in WorldCup



(c) MF for IReq in Clarknet (d) MF for IRep in Clarknet

Fig.9. Membership functions for different datasets.

Based on the optimal threshold values, the detection parameter values were divided into two sections. The values higher than the threshold fall in legitimate zone and values lower than the threshold fall in attack zone, as depicted in Fig. 7 and Fig. 8.

B. Membership Functions

The values of α and β are used in the Eq. (5) to grade the detection parameter inputs corresponding to every client. Based on the optimal threshold values calculated above, we compute the values of α and β for different datasets [41]. Fig. 9 shows the membership function (MF) values for each detection parameter and benchmark dataset.

C. Discussion

Fig. 7 and Fig. 8 represent the difference in the values obtained by legitimate users and bots during an attack. We plot boxes to represent the bot instances during different attacks in the figure. High rate attacks having the delay value of less than 300ms are easily identifiable by the detection parameters. However, the attacking bots involved in orchestrating asymmetric attacks are not detectable by *IReq* as the inter-request delay is very high. Requests in asymmetric attacks are generated at a very slow pace as compared to the other attacks.

The attacks (<H_100>, <H_200>) with 100ms and 200ms delay values are easily detected as the parameter values fall very below the optimal thresholds. The parameter values for the attack instance (<SP1_300>, <SP2_300>, and <SP3_300>) having delay value of 300ms fall nearly at the threshold edge, resulting in few false positives. As the bots randomly request for web pages from a large pool of resources in attacks like <H_100>, <H_200>, <H_300>, and <SP1_300>, *IRep* detection parameter classify these as legitimate instances. In case of asymmetric attacks (<AS_2500> and <AM_2500>), the values of *IRep* lie in the attack zone. This is due to the fact that single web page is repeatedly requested in <AS_2500> and a set of heavy sized web pages are repeatedly requested in <AM_2500> attack respectively. These attacks are not detectable by *IReq* but are easily visible using *IRep*.

The *low* and *high* demarcation of fuzzy values of *IReq* and *IRep* detection parameters is based on the threshold values computed during threshold optimization. Rule base that was applied by the inference engine of FCS to

Copyright © 2018 MECS

the inputs from fuzzification was also based on these threshold values. After receiving the results from FCS and scheduler, we observed a detection accuracy of 96.57% and 92.32%, for datasets WorldCup and Clarknet respectively. As a result, the average accuracy of our detection system becomes 94.45%.

Table 3. Comparison of proposed system with existing works

Works	DR (%)	FPR (%)	Attack strategy*
Liao Q *et al.* [42]	99.80	0.33	3
Yadav S *et al.*[43]	98.99	1.27	2
Wang J *et al.*[29]	88.95	5.10	2
Liao *et al.* [30]	89.25	0.04	4
Xie Y *et al.* [9]	90.00	1.00	1
Our work	94.45	0.97	5

*Based on the taxonomy proposed in our previous work

In comparison to previous existing works, the proposed system possesses the capability to identify a higher number of GET flood attack types while maintaining decent accuracy for both datasets (see Table 3). The number of attack types detected is calculated based on our previously designed taxonomy [11]. The values of Detection Rate (DR) and False Positive Rate (FPR) for other works were estimated after averaging the corresponding values given in the respective works.

VI. Conclusions

Modern day attackers make use of sophisticated bots that mimic human browsing behavior to orchestrate attacks on the application layer of the TCP/IP stack. Thus, it becomes cumbersome for the security mechanisms employed by the victim to discriminate between legitimate and bot clients. Therefore, it is imperative for an organization to employ strong protection systems in order to eradicate the effect of such attacks.

As these attacks mainly rely on bots behaving as legitimate clients, an effective detection is possible only if we monitor browsing behavior of each and every client over the time. This paper proposes a framework that employs uses two parameters to quantify the behavior of each client during a certain time window. These parameter values are then fed to our designed fuzzy control system that produces a legitimacy value for each client. These legitimacy values are used by the scheduler to filter out clients that score less than the designated threshold values. Our work is able to detect more number of GET flood attack types with 94.45% detection rate and 0.97 false positive rate. These results point toward the effectiveness of the proposed detection system in efficiently recognizing the presence of bots among the legitimate client. We plan to extend this work to detect more attack types by complementing the proposed system with more behavioral specific detection parameters.

References

[1] K. Kumar, R. C. Joshi, and K. Singh, "A Distributed Approach using Entropy to Detect DDoS Attacks in ISP Domain," In *Proceedings of the International Conference on Signal Processing, Communications and Networking*, 2007, pp. 331–337.

[2] "C2DF: High Rate DDOS filtering method in Cloud Computing - Semantic Scholar." [Online]. Available: /paper/C2DF-High-Rate-DDOS-filtering-method-in-Cloud-Shamsolmoali-Hamdard/5171336c8b0a5e4fb79cb5721f83ee72f28ffb36. [Accessed: 24-Feb-2017].

[3] A. Bhandari, A. L. Sangal, and K. Kumar, "Destination Address Entropy based Detection and Traceback Approach against Distributed Denial of Service Attacks," *Int. J. Comput. Netw. Inf. Secur.*, vol. 7, no. 8, pp. 9-20, Jul. 2015.

[4] "The 5 Most Significant DDoS Attacks of 2016," *The State of Security*, 29-Nov-2016. [Online]. Available: https://www.tripwire.com/state-of-security/security-data-protection/cyber-security/5-significant-ddos-attacks-2016/. [Accessed: 27-Feb-2017].

[5] K. Singh, P. Singh, and K. Kumar, "Impact analysis of application layer DDoS attacks: A simulation study," *Int. J. Intell. Eng. Informatics*, vol. 5, no. 1, pp. 80–100, 2017.

[6] K. Singh, P. Singh, and K. Kumar, "A systematic review of IP traceback schemes for denial of service attacks," *Comput. Secur.*, vol. 56, pp. 111–139, Feb. 2016.

[7] W. Zhou, W. Jia, S. Wen, Y. Xiang, and W. Zhou, "Detection and defense of application-layer DDoS attacks in backbone web traffic," *Future Gener. Comput. Syst.*, vol. 38, pp. 36–46, Sep. 2014.

[8] M. Srivatsa, A. Iyengar, J. Yin, and L. Liu, "Mitigating application-level denial of service attacks on Web servers: A client-transparent approach," *ACM Trans Web*, vol. 2, no. 3, pp. 15:1–15:49, Jul. 2008.

[9] Y. Xie and S.-Z. Yu, "Monitoring the application-layer DDoS attacks for popular websites," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 15–25, 2009.

[10] S. Lee, G. Kim, and S. Kim, "Sequence-order-independent network profiling for detecting application layer DDoS attacks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2011, no. 1, p. 50, Aug. 2011.

[11] K. Singh, P. Singh, and K. Kumar, "Application layer HTTP-GET flood DDoS attacks: research landscape and challenges," *Comput. Secur.*, vol. 65, pp. 344-372, Mar. 2017.

[12] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites," In *Proceedings of the International Conference on World Wide Web*, New York, USA, 2002, pp. 293–304.

[13] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using Hard AI Problems for Security," In *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed. Springer Berlin Heidelberg, 2003, pp. 294–311.

[14] S. Kandula, D. Katabi, M. Jacob, and A. Berger, "Botz-4-sale: surviving organized DDoS attacks that mimic flash crowds," In *Proceedings of the Symposium on Networked Systems Design & Implementation - Volume 2*, Berkeley, USA, 2005, pp. 287–300.

[15] W. Yen and M.-F. Lee, "Defending application DDoS with constraint random request attacks," In *Proceedings of the Asia-Pacific Conference on Communications*, 2005, pp. 620–624.

[16] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly, "DDoS-Resilient Scheduling to Counter Application Layer Attacks Under Imperfect Detection," In *Proceedings of the IEEE International Conference on Computer Communications*, 2006, pp. 1–13.

Copyright © 2018 MECS

[17] J. Yu, Z. Li, H. Chen, and X. Chen, "A Detection and Offense Mechanism to Defend Against Application Layer DDoS Attacks," In *Proceedings of the Third International Conference on Networking and Services*, 2007, pp. 54–54.

[18] G. Oikonomou and J. Mirkovic, "Modeling Human Behavior for Defense Against Flash-crowd Attacks," In *Proceedings of the IEEE International Conference on Communications*, Piscataway, USA, 2009, pp. 625–630.

[19] S. Wen, W. Jia, W. Zhou, W. Zhou, and C. Xu, "CALD: Surviving Various Application-Layer DDoS Attacks That Mimic Flash Crowd," In *Proceedings of the 4th International Conference on Network and System Security (NSS)*, 2010, pp. 247–254.

[20] D. Das, U. Sharma, and D. K. Bhattacharyya, "Detection of HTTP Flooding Attacks in Multiple Scenarios," In *Proceedings of the International Conference on Communication, Computing and Security*, NY, USA, 2011, pp. 517–522.

[21] S. B. Ankali and D. V. Ashoka, "Detection architecture of application layer DDoS attack for internet," *Int J Adv. Netw. Appl.*, vol. 3, no. 01, pp. 984–990, 2011.

[22] M. I. Ak, L. George, K. Govind, and S. Selvakumar, "Threshold Based Kernel Level HTTP Filter (TBHF) for DDoS Mitigation," *Int. J. Comput. Netw. Inf. Secur.*, vol. 4, no. 12, pp. 31-39, Nov. 2012.

[23] S. Limkar and R. K. Jha, "An Effective Defence Mechanism for Detection of DDoS Attack on Application Layer Based on Hidden Markov Model," In *Proceedings of the International Conference on Information Systems Design and Intelligent Applications*, 2012, pp. 943–950.

[24] S. Sivabalan and P. J. Radcliffe, "A novel framework to detect and block DDoS attack at the application layer," In *Proceedings of the IEEE TENCON Spring Conference*, 2013, pp. 578–582.

[25] J. Wang, M. Zhang, X. Yang, K. Long, and C. Zhou, "HTTP-sCAN: Detecting HTTP-flooding attaCk by modeling multi-features of web browsing behavior from noisy dataset," In *Proceedings of the 19th Asia-Pacific Conference on Communications (APCC)*, 2013, pp. 677–682.

[26] J. Wang, X. Yang, and K. Long, "Web DDoS Detection Schemes Based on Measuring User's Access Behavior with Large Deviation," In *Proceedings of the IEEE Global Telecommunications Conference* , 2011, pp. 1–5.

[27] L. C. Giralte, C. Conde, I. M. de Diego, and E. Cabello, "Detecting denial of service by modelling web-server behaviour," *Comput. Electr. Eng.*, vol. 39, no. 7, pp. 2252–2262, Oct. 2013.

[28] Y. Xie, S. Tang, Y. Xiang, and J. Hu, "Resisting Web Proxy-Based HTTP Attacks by Temporal and Spatial Locality Behavior," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1401–1410, 2013.

[29] J. Wang, X. Yang, M. Zhang, K. Long, and J. Xu, "HTTP-SoLDiER: An HTTP-flooding attack detection scheme with the large deviation principle," *Sci. China Inf. Sci.*, pp. 1–15, Apr. 2014.

[30] Q. Liao, H. Li, S. Kang, and C. Liu, "Application layer DDoS attack detection using cluster with label based on sparse vector decomposition and rhythm matching," *Secur. Commun. Networks*, vol. 8, no. 17, pp. 3111–3120, Nov. 2015.

[31] P. Xiao, W. Qu, H. Qi, and Z. Li, "Detecting DDoS attacks against data center with correlation analysis," *Comput. Commun.*, vol. 67, pp. 66 – 74, 2015.

[32] D. Kshirsagar and S. Kumar, "HTTP Flood Attack Detection Using Ontology," In *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, NY, USA, 2016, pp. 15:1–15:4.

[33] R. Kobayashi, G. Otani, T. Yoshida, and M. Kato, "Defense Method of HTTP GET Flood Attack by Adaptively Controlling Server Resources Depending on Different Attack Intensity," *J. Inf. Process.*, vol. 24, no. 5, pp. 802–815, 2016.

[34] T. Miu, C. Wang, D. X. Luo, and J. Wang, "Modeling User Browsing Activity for Application Layer DDoS Attack Detection," In *Proceedings of the International Conference on Security and Privacy in Communication Networks*, 2016, pp. 747–750.

[35] F. Pukelsheim, "The Three Sigma Rule," *Am. Stat.*, vol. 48, no. 2, pp. 88–91, 1994.

[36] Z. Hu, Y. V. Bodyanskiy, O. K. Tyshchenko, and V. M. Tkachov, "Fuzzy Clustering Data Arrays with Omitted Observations," *Int. J. Intell. Syst. Appl.*, vol. 9, no. 6, pp. 24–32, 2017.

[37] Z. Hu, Y. V. Bodyanskiy, O. K. Tyshchenko, and V. O. Samitova, "Fuzzy Clustering Data Given on the Ordinal Scale Based on Membership and Likelihood Functions Sharing," *Int. J. Intell. Syst. Appl.*, vol. 9, no. 2, pp. 1–9, 2017.

[38] L. Abdullah and A. Otheman, "A New Entropy Weight for Sub-Criteria in Interval Type-2 Fuzzy TOPSIS and Its Application," *Int. J. Intell. Syst. Appl.*, vol. 5, no. 2, p. 25, Jan. 2013.

[39] H. Beitollahi and G. Deconinck, "Tackling Application-layer DDoS Attacks," *Procedia Comput. Sci.*, vol. 10, pp. 432–441, Jan. 2012.

[40] H. Beitollahi and G. Deconinck, "ConnectionScore: a statistical technique to resist application-layer DDoS attacks," *J. Ambient Intell. Humaniz. Comput.*, vol. 5, no. 3, pp. 425–442, Jul. 2013.

[41] T. Chaira and A. K. Ray, "Threshold selection using fuzzy set theory," *Pattern Recognit. Lett.*, vol. 25, no. 8, pp. 865–874, Jun. 2004.

[42] Q. Liao, H. Li, S. Kang, and C. Liu, "Feature extraction and construction of application layer DDoS attack based on user behavior," In *Proceedings of the Chinese Control Conference (CCC)*, 2014, pp. 5492–5497.

[43] S. Yadav and S. Subramanian, "Detection of Application Layer DDoS attack by feature learning using Stacked AutoEncoder," In *Proceedings of the International Conference on Computational Techniques in Information and Communication Technologies*, 2016, pp. 361–366.

Authors' Profiles

Karanpreet Singh received the masters's degree in computer science and engineering, in 2013 and the bachelor's degree in information technology, in 2011 from Punjab Technical University, Jalandhar, Punjab, India. He is currently pursuing the Ph.D. degree in computer science and engineering at National Institute of Technology Jalandhar, Punjab, India. His research interests include network security, distributed networks, and cloud computing. He is a student member of the IEEE and the IEEE Communications Society.

Copyright © 2018 MECS

Paramvir Singh received the Ph.D. degree in computer science and engineering from Guru Nanak Dev University, Amritsar, Punjab, India, in 2011 and the M.Tech. degree in computer science and engineering from Panjab University Chandigarh, India, in 2005. He is currently with Department of Computer Science and Engineering, National Institute of Technology Jalandhar, Punjab. He has published more than 20 papers in refereed international journals and refereed international conferences proceedings. His research interests include software engineering, secure systems, and network security. He is a member of the IEEE and the IEEE Computer Society, and a life member of ISTE.

Krishan Kumar received the Ph.D. degree in electronics and computer engineering from Indian Institute of Technology, Roorkee, India. He is currently with the Department of Information Technology, University Institute of Engineering and Technology, Panjab University, Chandigarh, India. His research interests include network security, network measurement/modeling, manets and WSNs. He has published more than 70 papers in refereed international journals and conference proceedings. He is on editorial board of many reputed international journal and conferences in the field of networking.

How to cite this paper: Karanpreet Singh, Paramvir Singh, Krishan Kumar, "Fuzzy-based User Behavior Characterization to Detect HTTP-GET Flood Attacks", International Journal of Intelligent Systems and Applications(IJISA), Vol.10, No.4, pp.29-40, 2018. DOI: 10.5815/ijisa.2018.04.04

Copyright © 2018 MECS