User behavior analytics-based classification of application layer HTTP-GET flood attacks

Karanpreet Singh1\*, Paramvir Singh1, Krishan Kumar2

1 Department of Computer Science and Engineering, NIT Jalandhar, India

2 Department of Information Technology, UIET, Panjab University, Chandigarh, India

[*\*karanpreet.13.cse@nitj.ac.in*](mailto:*karanpreet.13.cse@nitj.ac.in)

Abstract

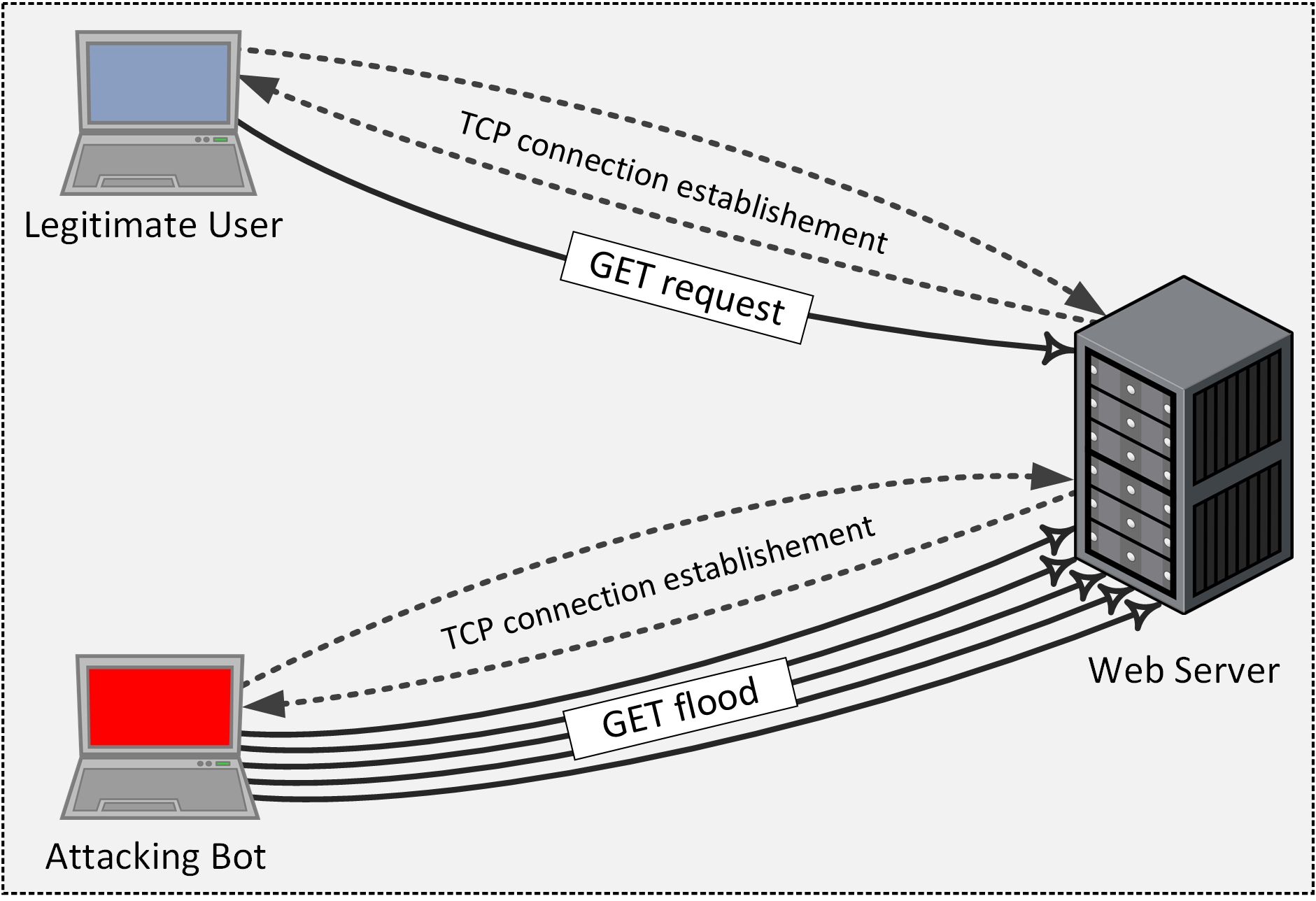
Web services are one of the most prominent forms of web presence exercised by the businesses to connect to their possible client base. GET flood attacks, commonly known as Application Layer DDoS attacks, are widely executed exploits that challenge almost all the web servers hosting such services on the Internet. The state-of-art literature provides many security mechanisms that are designed to handle such attacks, however, attackers constantly explore unique approaches for orchestrating covert GET flood attacks. The detection of such attacks requires user level monitoring due to a high resemblance among the browsing behaviors of legitimate users and modern-day sophisticated bots. In this paper, we propose four novel behavioral features to distinguish GET flood attack sources from the legitimate normal and flash traffic. Our work distinguishes itself from previous works by providing a comprehensive solution for the detection of 12 different strategies employed by an attacker to launch GET flood attacks. We build an experimental test bed supported by well-known software tools that replay the benchmark web logs and generate emulated attack traces pertaining to GET flood attack strategies. The datasets prepared during the course of this experimentation are evaluated through an exhaustive performance comparison of the selected set of machine learning classifiers. The obtained results evidently indicate significantly high detection accuracy (97.46%) with few false alarms using the SVM classifier.

Keywords: *denial of service, application layer attack, GET flooding, intrusion detection*

1. Introduction

Distributed Denial of Service (DDoS) attacks have been a prominent threat to the availability of online services despite the presence of numerous security measures deployed at different levels of the Internet. It remains an open research issue as the efforts of the security community are constantly challenged by the rising sophistication of the attackers (Cui *et al.*, 2016). The companies using the internet as their business platform rely on web-based applications to deliver a variety of services and connect with their consumers. However, the expanding realm of these web services has captivated many unethical entities that seek to vitiate the service available to the intended user base. The compromised systems that an attacker exploits for launching an attack are known as bots. These bots are responsible to send huge volumes of unsolicited traffic toward the victim (Chen *et al.*, 2007).

Attackers have progressively exercised significant functional up-gradations on these bots in order to carry out stealthier attacks. As per a recent report (*Bad Bot Landscape Report*, 2016), the sophistication of bots is rising in terms of their behavioral proximity with human browsing characteristics. Using legitimate IP addresses (Singh *et al.*, 2016), sophisticated bots are able to process JavaScript, follow graphical hyperlinks, and perform TCP connection establishment procedures, enough to trick any server into perceiving the source as legitimate human. Out of the total bots identified in the year 2015, 46% were advanced bots. 53% of bots are able to deal with complex logical web elements and 39% are able to mimic humans. Attacker nowadays uses these sophisticated bots to portray a false resource demand by making them send multiple requests to overload the server, as shown in Fig. 1. Consequently, the request queue of a server becomes clogged leading to isolation of subsequent legitimate requests (Singh *et al.*, 2017a). Such an attack routine is termed as HTTP-GET flood DDoS attack as it inundates an HTTP based application using a deluge of GET requests.



**Fig. 1.** Sophisticated bot launching a GET flood DDoS attack

During recent years, web servers have witnessed countless DoS attempts at the application layer. According to a report by Arbor Networks (*Worldwide Infrastructure Security Report*, 2017) published in Q1 of the year 2017, 80 percent of their respondents have reported seeing application layer DDoS attacks targeting DNS and HTTP services. Moreover, HTTP has been the top targeted application layer service. The dominance of sophisticated bots, capable of mimicking the unique characteristics associated with the legitimate browsing behavior, allows the GET flood attack requests sneak through the traditional attack detection mechanisms. To exacerbate the situation, attackers have devised numerous complex strategies to execute GET flood attacks. Our previous review study (Singh *et al.*, 2017b) presents and explains such GET flood attack strategies that have been studied in the literature.

An efficient detection of GET flood attacks requires characterization of legitimate and GET flood attack traffic at individual user level at the server end. In order to facilitate such user-level traffic inspection, this study proposes a feature set comprising four behavior specific features. These four features take advantage of one or more browsing related activities of the legitimate users and attack bots to provide detection solution capable of identifying bots during normal and flash background traffic. Our work distinguishes itself from other works by providing a comprehensive solution for the detection of 12 different strategies employed by an attacker to launch GET flood attacks.

To the best of our knowledge, there are no benchmark traffic traces of GET flood attacks. Although there are daily instances of GET flood attacks, the reluctance of the victim organizations to publicize their web logs is the primary reason for such scenario. Therefore, the researchers use various attack tools to generate the respective traffic traces. These attack traffic traces are then mixed with the attack-free benchmark traffic traces. Many recent studies (Beitollahi and Deconinck, 2013; Huang *et al.*, 2014; Jazi *et al.*, 2017; Liao *et al.*, 2015) have employed such a practice and used the freely available benchmark traffic traces as carriers of their synthesized attack traces. We design and build an experimental setup on top of DDoSTB (Behal and Kumar, 2016), and emulate user and bot clouds that generate legitimate and attack traffic respectively to reproduce several access logs. The legitimacy of these attack datasets is improved by reproducing the original server file structure on our emulated server system. Different type of GET flood attack traffic is then mimicked by tuning the bot parameters using a set of well-known software tools. These access logs are processed for the feature values to build multiple datasets, which are used in training and testing six classifier models. The major contributions of this work are as follows.

* Multiple emulated scenarios built on DDoSTB (Behal and Kumar, 2016) to fabricate different GET flood attack strategies on a carefully designed experimental setup using well-known software tools and benchmark web logs.
* Characterization of legitimate and attack bots at individual user levels to assist network administrators in carrying out necessary filtering of the attack traffic.
* A minimal feature set comprising four behavior-inspired features for an efficient detection of 12 different types of GET flood attacks during normal and flash background traffic.
* Validation of the proposed feature set by evaluating and comparing six machine learning classifier algorithms using multiple performance parameters.

This paper is organized as follows. Section 2 presents the related work on detection of GET flood attacks. Section 3 introduces the proposed detection model comprising features and the undertaken machine learning classifiers. It also outlines various GET flood attack strategies that have been studied under this work. Section 4 details out each of the phases constituting the design of our complete experimental setup. Section 5 discusses the complexity associated with the proposed system along with outlining the different deployment options. Section 6 examines the results obtained from the experiments. Finally, Section 7 concludes the paper and discusses future research avenues.

1. Related Work

GET flood DDoS attacks target web services running on the application layer of the network stack. Consequently, these attacks have often been referred as *application-layer DDoS attacks* in the present literature (Lee *et al.*, 2011; Xie and Yu, 2009b; Zhou *et al.*, 2014). The rising sophistication of GET flood attacks requires continuous research efforts toward shielding the availability of web services. The present literature mainly focuses on detecting GET flood attacks by capturing anomalies in the behavior of individual users during the course of their browsing activities. Nonetheless, many other works perform entry-level user inspections using challenge-based techniques to avoid admitting malicious elements into the system. The existing literature is discussed below in chronological order.

In 2009, Xie and Yu (2009a) proposed an approach to model the sequence order of legitimate page requests and characterize the legitimate and suspicious browsing behavior based on Hidden semi Markov Model (HsMM). The deviation of entropy from the defined threshold limits identifies the presence of bots. The authors assumed that the bots do not follow the hyperlinks present on a webpage; instead, they use URLs to access the new webpages. This assumption may not hold in case of stealthy bots as they can easily follow hyperlinks. Moreover, the proposed algorithm demands substantial computational resources for real-time detection. Yu *et al.* (2010) proposed a trust-based system to prioritize the requests received by the server. Depending on the user’s connection history, four different trust values are assigned to the user; long-term trust, short-term trust, misusing trust and negative trust. These values collectively known as a license are stored on the client side in the form of cookies. The trust-based scheduler then decides to accept or reject the connection based on the values received in a license provided by the user.

In 2010, Du and Nakao (2010) proposed a credit based attack detection system called OverCourt. Every user is assigned a credit value based on the amount of packets exchanged with the server. The system punishes the users that deviate from normal behavior by lowering their credit points while allowing well-behaving users to migrate over the protected communication channels. This scheme, however, requires maintaining a per-flow state of its users, which might cause an overhead on the server itself. Xuan *et al.* (2010) proposed a statistical detection approach based on group testing. A group of users is tested as a whole for its abnormality. The average response time of the requests is used to evaluate the group test result, which is either positive or negative. This decides the probability of the presence of suspicious elements in a group. However, a group testing based scheme will also punish legitimate users present in a suspected group.

In 2011, Choi *et al.* (2011) proposed a detection mechanism that used support vector machine to classify normal and attack traffic. The detection is based on the traffic characteristics collected during a specified monitoring period. This monitoring period is divided into a number of timeslots during which only a single HTTP GET request is allowed to be served. The normal and attack profiles are modeled using parameters extracted from each timeslot in a single monitoring period.

In 2012, Yu *et al.* (2012) assumed the network traffic to possess a strong similarity/correlation. They used correlation coefficient to differentiate a flash crowd from application-layer DDoS attacks. However, this system overloads the server by introducing complex computational effort, which bound its implementation in real time scenarios. Ye *et al.* (2012) proposed a time and sequence independent hierarchical clustering based detection scheme to differentiate legitimate and suspicious browsing behaviors. They used four different user session features: object size, request rate, object popularity and transition probability. This detection technique fails to identify attack traces in case of flash crowds.

In 2013, Ni *et al.* (2013) proposed a detection mechanism based on the entropy of HTTP GET requests per source IP. It utilized the fact that the source IP clusters are more distributed in case of the flash crowd as compared to a DDoS attack. It can differentiate flash crowd from possible application-layer DDoS attacks. Kalman filter is used to model various time-dependent parameters associated with an adaptive autoregressive model. The HTTP GET requests are classified using SVM (Support Vector Machine) trained by AAR (Adaptive Auto-Regressive) parameters. The adaptive behavior of the system allows the detection mechanism to work even in case of varying traffic conditions. Giralte *et al.* (2013) represented the legitimate user behavior in terms of layer 4 and layer 7 parameters like number of GET requests, GETs mean, mean of flows per user, standard deviation of flows per user, etc. A three-stage model was designed to detect a variety of application-layer DDoS attacks. The proposed scheme is able to distinguish legitimate web bots and attacking web bots based on their access path patterns. Xie *et al.* (2013) proposed a scheme that primarily detects web proxy based DDoS attacks using HsMM. The authors captured temporal and spatial localities to model web proxies’ access behavior using the server logs. The scheme offers traffic intensity and web content independent defense approach against proxy based attacks. However, with an increase in the number of users, the model is likely to give expensive results.

The popularity of a large website varies with time, as contents are regularly updated and deleted. In 2014, Wang *et al.* (2014) proposed a dynamic popularity based DDoS detection scheme based on their previous work (Wang *et al.*, 2011). Large deviation principle characterizes the difference in expected and actual popularity of webpages. The proposed system efficiently detects random and perfect knowledge DDoS attacks but is inadequate in defending single-URL and multi-URL DDoS attacks. Xu *et al.* (2014) proposed a scheme to detect asymmetric application-layer DDoS attacks. They captured user browsing sequence patterns based on extended random walk model. The proposed scheme predicts the possible future request sequence for a user based on the legitimate request sequence model. The scheme is able to identify individual attacker based on its deviation from the expected behavior. However, its restrained ability in detecting attacks based on high workload requests only makes it vulnerable to other types of application-layer DDoS attacks.

In 2015, Liao *et al.* (2015) proposed a machine learning-based detection technique that used support vector machine (SVM) to identify the presence of attacks. The similarity among access patterns of bots is measured using the request-frequency sequence feature. Subsequently, the rhythm-matching algorithm is applied to identify similar patterns. New and stealthy bots can easily evade such similarity-based detection techniques. Xiao *et al.* (2015) proposed a detection technique based on the property that the traffic flows generated by bots are likely to be correlated with each other. K-nearest neighbors algorithm is used to identify similarity among flows. Depending on this similarity among flows, the spurious access patterns were identified. Relying only on the similarity of flows does not always guarantee optimum detection accuracy.

In 2016, Sree and Bhanu (2016) proposed to detect GET flood attacks through a hierarchical processing of web logs and using Dempster-Shafer theory of evidence to identify spurious users. The proposed system induced minimal processing time for profile construction from web logs to provide high accuracy and low false alarms. Only a few number of GET flood attack strategies are evaluated for detection by this system. Kshirsagar and Kumar (2016) provided an ontology of HTTP requests to assist detection of GET flood attacks. However, sophisticated bots with the capability of producing request similar to a legitimate user can easily evade such ontology-based detection systems. Miu *et al.* (2016) employ the browsing behavior to identify anomalous users. They monitored the sequence of web page access by the users. Based on the expected transition probability between the web pages, the browsing behavior of every user is quantified. The log likelihood of a session is taken as an attribute to differentiate bots and legitimate users. Sophisticated bots can easily circumvent this detection technique. Yadav *et al.* (2016) used AutoEncoder, a deep learning neural network model, to extract prominent traffic features from the attack datasets. A logistic regression classifier is then applied to the extracted features to detect various GET flood attacks. This work does not highlight any details about the fabrication of attacks datasets. Moreover, only a few number of GET flood attack strategies are taken into consideration.

In 2017, Zhang *et al.* (2016) proposed a challenge based defense system known as SENTRY. Their system moderates the user requests before forwarding it to the server. This significantly reduces the load on the server. Being a challenge-response based approach, the effective turnaround times of the user requests will be more than the normal. Moreover, deployment of such a technique in a high workload environment with a large number of users can really be challenging. Jazi *et al.* (2017) made use of both network and application layer metrics to detect application layer DDoS attacks. Non-parametric CUSUM algorithm is used to detect changes in the proportions of the network packets without payloads and packets carrying the HTTP service requests. They further studied the impact of various sampling techniques on the detection performance of the system. Although low rate attacks like slow send, slow read, etc. can be detected but low-rate high-workload intensive, i.e. asymmetric GET flood attacks might evade their detection mechanism.

A systematic classification of state-of-the-art literature on GET flood DDoS attack detection is presented in our previous survey work (Singh *et al.*, 2017b). It categorizes a number of detection attributes that have been utilized in the related works. These also include various user and traffic specific features that cater to GET flood DDoS attack detection when supported by machine learning algorithms. In this paper, we introduce four detection features that can effectively distinguish between legitimate users and bots by capturing individual users’ browsing semantics from the incoming traffic. In contrast to existing features, this minimal set of four proposed features offers highly accurate detection of bots responsible behind 12 GET flood attack strategies defined in the literature.

1. Proposed Detection Model

The sophisticated bots employed to carry out GET flood attacks can successfully imitate traits of legitimate users. As a result, the task of differentiating a bot from a legitimate user at the server end becomes strenuous. An in-depth examination of browsing behaviors of bots and legitimate users, therefore, become necessary to derive inimitable characteristics that allow discrimination between the two. This study utilizes such behavioral characteristics to propose four features that can expose bots hiding behind the identity of legitimate users. Labelled datasets are prepared by extracting these four features from the emulated access logs. The complete process of preparing datasets (including web log cleaning, preliminary analysis, and feature calculation) is discussed in Section 4. The primary goal of this work is to train various machine learning classifiers and compare their classification efficiency using these datasets.

* 1. Feature definitions

Our detection model operates on two time windows Tn (narrow) and Tw (wide) with a time span of 30 seconds and 120 seconds respectively. Every single wide time window Tw is composed of four consecutive narrow time windows Tn. The feature values for each client are computed at the end of every wide time window Tw i.e. after every 120 seconds. Thus, the delay in detection of the bots for an on-going attack will also be 120 seconds. As the wide windows of 120 seconds are composed of four 30-second windows, lowering the duration of narrow time windows significantly reduces the detection performance. Apparently, the proposed system with longer length time windows is more efficient in terms of detection accuracy. The duration of time windows taken in this study has been finalized based on the trade-off between the time required for detection and the detection accuracy.

Various initial parameters (transition frequency *α*, response size distribution *ƥ*, kernel function *h,* etc.) required for feature computation are evaluated during the preliminary analysis phase discussed in Section 4. It may be noted that these values are unique for each benchmark web log used in this study. The proposed features are defined below.

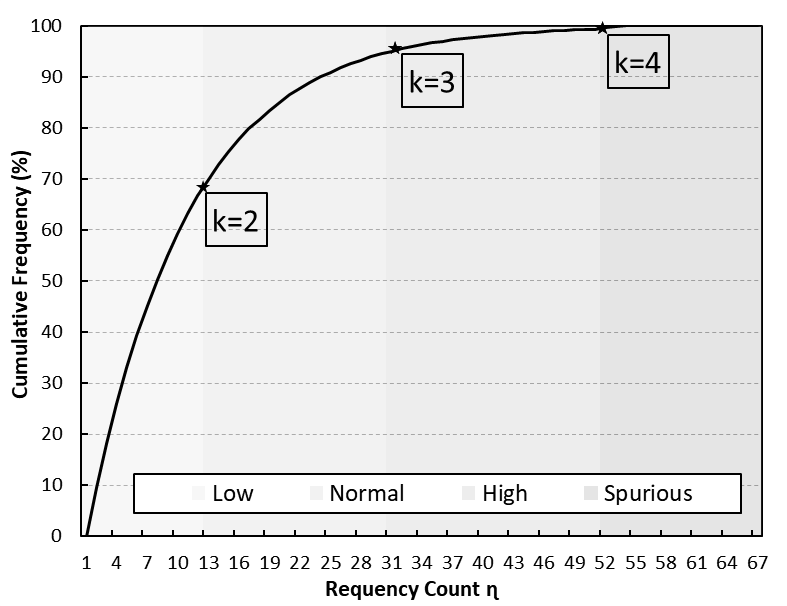
|  |
| --- |
|  |
| **(a)** Legitimate user |
|  |
| **(b)** Attacking bot |
| **Fig. 2.** Difference in browsing behavior of legitimate users and attacking bots |

* + 1. Request Index

Previous studies (Bhandari *et al.*, 2016; Yu *et al.*, 2015) have stated that the legitimate users are fairly large in number as compared to the active bots (except superbots) taking part in an attack. Therefore, it becomes obligatory for an attacker to raise the frequency of requests generated by the bots in order to make an impact on the server's performance. The request rates of bots will inevitably be higher than that of the legitimate users. A user browsing sequence is divided into up-time and down-time. *Up-time* denotes the time period in which a user makes requests to the server.

The amount of time a user spends viewing the requested resources is termed as *down-time*. Based on our analysis of legitimate and bot browsing behaviors, we believe that a typical user browsing pattern consists of repetitive cycles of two activity sets (S1 and S2). The activity set S1 defines users that request and view a single resource from the server. In contrast, the activity set S2 defines users that issue simultaneous requests for multiple (more than one) server resources. Once all the requested resources are received, a user then spends time (down-time) to view them sequentially. A user makes the transition to either same activity set or to the other activity set, as shown in Fig. 2. The complete browsing history of a user can therefore be represented as a series of these activity sets. It should be noted that the legitimate users spend a significant amount of time viewing (down-time intervals) the requested web pages (Fig. 2a) whereas bots, with an intention to overload the server, quickly issue further requests without extending their down-times (Fig. 2b). Consequently, the request rates of the legitimate users are much lower than that of the attack bots.

Using these assumptions, we compute the value of Feature *Ƒ*1 as follows. We divide the range of request rates (*η*) into four classes namely Low (L), Normal (N), High (H), and Spurious (S), constituting the set *Ɍ*. We use Chebyshev’s Inequality theorem (Pukelsheim, 1994) to partition the skewed distribution of request frequency range. It states that at least observations are within *k* standard deviations of the mean. The distribution is divided into four classes using three percentages 75%, 89%, and 93.7% with 2, 3, and 4 (number of standard deviations ‘k’) standard deviations at respectively. Fig. 3 shows the distribution of request frequencies into different classes. Let us assume that the range of request rates is divided as follows.



**Fig. 3.** Mapping request frequencies to class using Chebyshev theorem

|  |  |  |
| --- | --- | --- |
|  |  |  |

These limits are used to map the requests made by a particular user in consecutive time windows to their respective classes. For instance, consider an event where a user sends 15, 36, 4, and 61 numbers of requests in four consecutive narrow time windows, respectively. Then the request class sequence of this user becomes N → H → L → S.

The above set of sequence contains three transitions among request classes i.e., normal to high, high to low, and low to spurious. Similarly, there exist 12 such transition possibilities among 4 request classes. This process is repeated for every user in the benchmark web logs during out preliminary analysis phase to calculate the frequency of each of these 12 class transitions. Using these pre-processed values, a score value (congruent to their relative probabilities) using Eqn. (1) is assigned to each of the 12 class transitions.

|  |  |
| --- | --- |
|  | (1) |
|  |  |
|  | | (2) |
|  | |  | |

*λ* is known as the relaxing factor and is computed using Eqn. (2). *α* represents the frequency of transitions (in benchmark web logs) from class *x* to class *y* and represents the score value of transition from class *x* to class *y*.

Request Index (*Ƒ1*)is computed for every connected user using Eqn. (3) as the averaged sum of transition scores for consecutive time windows *Tn* in a single *Tw*.

|  |  |
| --- | --- |
|  | (3) |
|  |  |

maps a particulate request frequency value to its respective class. *η* represents the number of requests in time window *T*. is the number of narrow windows in a single *Tw*.

* + 1. Response Index

The primary aim of an attacker is to make the server inaccessible to the legitimate users by keeping it busy in processing requests received from bots. Realizing it requires the bots to either generate requests at a very high rate or request for large sized (in bytes) resources. The former can be identified using *Ƒ1* whereas, in order to identify the latter, we propose a feature namely Response Index (*Ƒ2*), whose value depends on the amount of data a user requests from the server in a single time window.

We initially compute the response size distribution (*ƥ*), which corresponds to the amount of data (bytes) requested by individual users during a single time window *Tn*. According to adjusted box plot (Hubert and Vandervieren, 2008), the upper boundary of this distribution is estimated using Eqn. (6)

|  |  |
| --- | --- |
|  | (4) |

are independent samples from the distribution *ƥ,* is the median of *ƥ* and *h* is the kernel function estimated as

|  |  |
| --- | --- |
|  | (5) |

|  |  |
| --- | --- |
|  | (6) |

Q3 and IQR are the third quartile and interquartile range of the distribution, respectively. MC is the med couple. *Ƒ2*is computed using Eqn. (7).

|  |  |
| --- | --- |
|  | (7) |

and represent the aggregated response size and response upper bound, respectively.

* + 1. Popularity Index

### As stated in many studies (Xie and Yu, 2009a; Ye and Zheng, 2011), almost 10% of the web pages receive 90% of the total incoming traffic. These web pages (popular webpages) are collectively known as *hot set*. Legitimate users often tend to request for web pages belonging to this set. Assuming the web pages associated with *hot set* are unknown to an attacker, the request targets of bots are likely to be randomly distributed. The proposed feature *Ƒ*4 utilizes this assumption to scores individual user using Eqn. (8).

|  |  |
| --- | --- |
|  | (8) |

is 0.9, is 0.1; is computed using Eqn. (9), = 1 – ; and ɛ is a very small value used to exceptions in cases where values become zero.

|  |  |
| --- | --- |
|  | (9) |

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

represents the set of web pages requested in time window *Tn*.

* + 1. Repetition Index

### The legitimate users rarely repeated their requests for same web resource in a particular time window i.e., an already visited web page is not requested again soon. Whereas the bots demonstrating a random access behavior are more likely to re-visit a web resource. Moreover, due to their higher request rates, this re-visitation probability rises. Feature *Ƒ*4 to capture such instances using Eqn. (11).

|  |  |
| --- | --- |
|  | (11) |

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

represents the set of web pages requested in time window *Tn.*

* 1. Classifier algorithms

The prime objective of the proposed system is to provide a classification model capable of efficiently discriminating legitimate users from the bots attempting HTTP-GET flood attacks. The selection of a suitable machine learning classification algorithm, therefore, becomes vital. Machine learning classification algorithms can be grouped into supervised and unsupervised sets. Supervised machine learning algorithms aim at categorizing to a class the unseen input samples based on the modeled training dataset with known classes. In contrast, unsupervised machine learning algorithms cluster the unlabeled feature space and classify the new sample based on its proximity to existing clusters. This study considers comparing the classification performance of six supervised machine learning classifiers: Naïve Bayes, Support Vector Machine (SVM), Random Forest, J48, JRip, and IBk, which are known to offers decent accuracy levels with low computation costs. These classifiers are briefed below.

1. *Naïve Bayes:* Naïve Bayes classifier works on the principle of Bayes theorem. Irrespective of feature independence constraint, its applicability still persists in a wide variety of application domains due to high accuracy and low training time.
2. *J48:* J48 is a decision tree JAVA implementation of the C4.5 algorithm. It is used to compute different case sets to classify an instance. These case sets are represented in the form of a tree that carries tests on the internal nodes to determine the path toward a leaf assigned with a unique class label.
3. *JRIP:* JRIP implements RIPPER, a rule-based classification taking into consideration reduced error pruning. It offers rapid rule generation while maintaining the expressiveness close to decision trees.
4. *SVM:* We incorporated the package LibSVM (Chang and Lin, 2011) for experimenting with Support Vector Machine (SVM) in Weka. SVM can effectively operate on high dimensional feature space by partitioning the training data into hyperplanes with maximal margins. The training data is mapped onto the higher dimensional space using various kernel methods in an attempt to form separable and structured feature space. It is one of the widely implemented classifiers due to its high accuracy for large-sized training dataset.
5. *Random Forest:* Random Forest follows an ensemble approach wherein an unknown instance is fed to a number of decision trees (collectively known as forest) for classification. The highest chosen label among these trees defines the final class of that instance. Much like SVM, it also handles large datasets reasonably well.
6. *IBk:* Weka provides an implementation of k-Nearest Neighbor (kNN) algorithm in form of Instance-Based Learner (IBk). The prediction of an unknown instance takes place using neighbor votes based on their respective distance measures. It performs an in-time prediction without constructing any classification model in advance, thus referred as a lazy classifier.
   1. Benchmark web logs

Every user access received by the server is usually registered as separate record entries in a file known as web logs (Wang *et al.*, 2005). Although web logs are privately maintained by server administrators, there exist several web logs that are publicly accessible on the Internet. These logs, commonly known as benchmark web logs, provide researchers a common platform for the evaluation of their respective proposed works.

In this study, we took three benchmark web logs that include WorldCup98, Clarknet, and NASA, to investigate characteristics pertaining to users' browsing behavior. The selection of these benchmark web logs is based on our previous literature assessment (Singh *et al.*, 2017b) according to which these three are the most used traffic traces. These web logs have been used in many high-quality works (Behal and Kumar, 2017; Bhatia *et al.*, 2014; Xie and Yu, 2009b; Yu *et al.*, 2012). The details of these web logs are as follows.

1. *WorldCup98*: The complete traffic trace of WorldCup98 spans over a period of 92 days, i.e. April 30, 1998 to July 26, 1998. Out of the entire web log, we took a two-hour segment with 956898 records (10509 unique users) from 42nd day to represent legitimate traffic characteristics. Besides this, we purposely took a 10-minute trace from the 66th day to quantitatively assess our proposed set of features during a Flash Event (FE).
2. *Clarknet*: The traffic trace of the first day is extracted from the dataset of seven days (August 28, 1995 to September 3, 1995) for the identification of legitimate user behavior. 251334 records (22569 unique users) of user activities from this web log are handled during the pre-processing stage.
3. *NASA*: This web log is considerably small as compared to the WorldCup98 and Clarknet web log. A full day traffic trace of 1st July, 1995 containing 64714 records (5129 unique users) is taken into consideration from NASA WWW access logs.
4. In addition to the above-mentioned benchmark web logs, we decided to incorporate in the experiment the user accesses registered within two consecutive days by our university website[[1]](#footnote-1). This access log contains 80198 records belonging to 1051 unique users captured from June 28, 2016 to June 29, 2016. The recording started the day admission announcements were put into view on the university website in order to acquire access behaviors of a variety of different users.
   1. HTTP-GET flood attack strategies

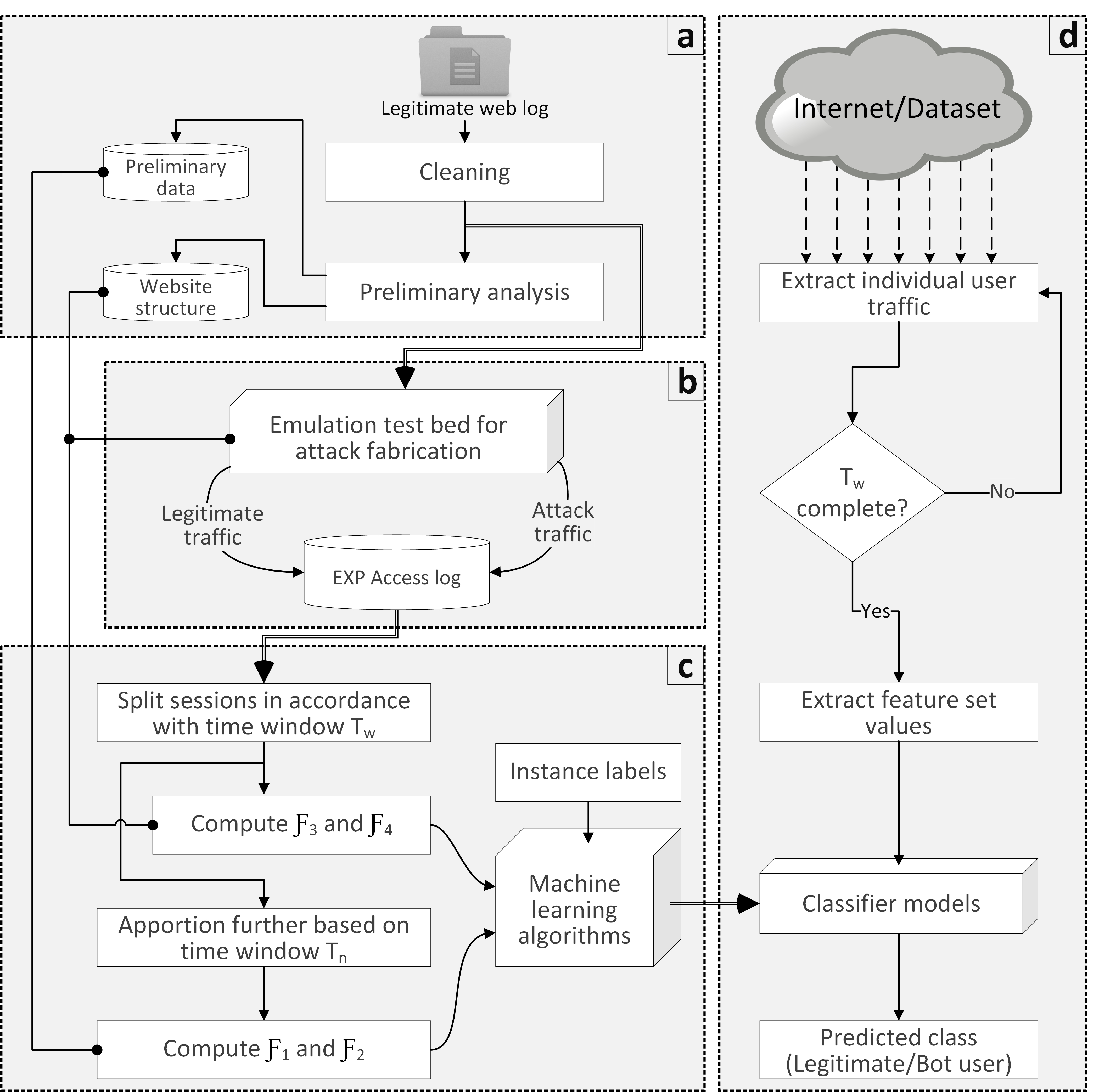
Attackers follow a number of complex maneuvers to perform GET flood attacks. A comprehensive classification of various possible GET flood attack strategies is given in our previous study (Singh *et al.*, 2017b). In this paper, we consider the detection of 12 commonly studied GET flood attack strategies. These attack strategies are generated by varying five configuration parameters namely delay, selection, count, sequence, and volume. *Delay* refers to the time gap between two consecutive requests; *selection* refers to the set of web pages on server requested by bots; *count* refers to the number of unique web pages requested; *sequence* refers to the order of requested web pages; *volume* refers to the range of web page sizes (bytes). These attack strategies are represented as "Attack Strategy (Category hierarchy)", and are elaborated below.

1. *Random (Server Load, High Rate):* The bots generate a continuous burst of requests to produce a load close to the server threshold. The load fluctuates either above or below this threshold.
2. *Flash (Server Load, High Rate):* This attack strategy is also known as *burst attacks*. This attack attempts to reproduce the effect of a flash event, thus making the distinction between authentic flash event and attack cumbersome. The request rate in this attack is significantly above the server threshold.
3. *Constant (Server Load, High Rate):* A major portion of literature has worked on the detection of this type of attack strategy. All the bots involved in this attack maintain a consistent request rate throughout the attack. Usually, the selected rate is either 100, 200, or 300 requests per second.
4. *Main Page (Single, Target Webpages, High Rate):* The bots usually target main page or home page of a website. The request rate is considerably higher than normal request rate.
5. *Dominant Page (Single, Target Webpages, High Rate):* A web page that is popular among the legitimate users is usually selected as the target of this attack.
6. *Repeated (Single, Target Webpages, High Rate):* The sequence of requested web pages is continuously repeated by the bots. This sequence is either pre-established or chosen at random.
7. *Replay Flood (Multiple, Target Webpages, High Rate):* Bots mimic the request patterns of legitimate users by capturing their access patterns then replaying the same at an inflated rate.
8. *Random Page (Multiple, Target Webpages, High Rate):* Some websites group their webpages into different categories such as sports, entertainment etc. In this attack, the bots follow hyperlinks to requests for web pages of randomly selected categories.
9. *Hot Pages (Page Interest, Target Webpages, High Rate):* The bots target only those web pages that belong to the set *Ɍ*. This attack is difficult to detect due to its inherent similarity with legitimate browsing behavior.
10. *Session Flood (High Rate):* Repeated sessions are generated by the bots in very less time to overload the server. A single bot maintains multiple sets of sessions at any particular instant.
11. *High Burst (Periodic, Symmetric, Low Rate):* The amplitude of an attack pulse generated in this strategy is lower than that of the high burst attack strategy. These attacks do not however affect the performance of high-end websites but could definitely deteriorate the functioning of low-end websites.
12. *Continuous (Asymmetric, Low Rate):* Workload intensive requests are generated by the bots at a slow pace. A single request is able to initiate recursive operations on the server.

The proposed system has not been tested against the attack strategies such as *Low Burst* (Periodic, Symmetric, Low Rate), *Non Periodic* (Symmetric, Low Rate), *Slowloris* (Symmetric, Low Rate), *One Shot* (Asymmetric, Low Rate), *Rare and Frequent Change* (Page Interest, Target Webpages, High Rate), *Web Proxy* (High Rate). The execution logic behind these attacks demands specialized detection solutions. These attacks have not been much prevalent in the literature concerning GET flood attacks. Therefore, we decided to avoid complicating the proposed system by incorporating support for their detection.

**Table 1** Parameters defining GET flood attack strategies and corresponding feature set fluctuations

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Attack strategy | Selection | Count | Sequence | Size | Delay δ (ms) | Ƒ1 | Ƒ2 | Ƒ3 | Ƒ4 |
| 1. High rate |  |  |  |  |  |  |  |  |  |
| 1.1. Server Load |  |  |  |  |  |  |  |  |  |
| 1.1.1. Random | Random | Random | Random | Random | 100 ≤ δ ≤ 1000 | ↑ | ↑ | ↕ | ↓ |
| 1.1.2. Flash | Random | Fixed | Random | Random | 100 ≤ δ ≤ 1000 | ↑ | ↑ | ↑ | ↑ |
| 1.1.3. Constant | Random | Random | Random | Random | 100 ≤ δ ≤ 1000 | ↑ | ↑ | ↑ | ↓ |
| 1.2. Target Webpages |  |  |  |  |  |  |  |  |  |
| 1.2.1. Single |  |  |  |  |  |  |  |  |  |
| 1.2.1.1. Main page | Popular | Single | - | Random | 100 ≤ δ ≤ 1000 | ↑ | ↕ | ↑ | ↕ |
| 1.2.1.2. Dominant | Popular | Single | - | Random | 100 ≤ δ ≤ 1000 | ↑ | ↕ | ↑ | ↕ |
| 1.2.2. Multiple |  |  |  |  |  |  |  |  |  |
| 1.2.2.1. Repeated | Random | Fixed | Fixed | Random | 100 ≤ δ ≤ 1000 | ↑ | ↑ | ↑ | ↕ |
| 1.2.2.2. Replay flood | Random | Fixed | Fixed | Random | 100 ≤ δ ≤ 1000 | ↕ | ↕ | ↑ | ↕ |
| 1.2.2.3. Random | Random | Random | Random | Random | 100 ≤ δ ≤ 1000 | ↑ | ↑ | ↕ | ↑ |
| 1.2.2.4. Page interest |  |  |  |  |  |  |  |  |  |
| 1.2.2.4.1. Hot page | Popular | Fixed | Random | Random | 100 ≤ δ ≤ 1000 | ↑ | ↑ | ↑ | ↕ |
| 1.2.2. Session flood | Random | Random | Random | Random | 100 ≤ δ ≤ 1000 | ↑ | ↑ | ↑ | ↕ |
| 2. Low Rate |  |  |  |  |  |  |  |  |  |
| 2.1. Symmetric |  |  |  |  |  |  |  |  |  |
| 2.1.1. Periodic |  |  |  |  |  |  |  |  |  |
| 2.1.1.1. High burst | Random | Random | Random | Random | 100 ≤ δ ≤ 2000 | ↕ | ↑ | ↕ | ↑ |
| 2.2. Asymmetric |  |  |  |  |  |  |  |  |  |
| 2.2.1. Continuous | Random | Fixed | Random | Large | 2000 ≤ δ ≤ 4000 | ↕ | ↑ | ↑ | ↓ |

****

**Fig. 4.** Experimental Setup

**(a)** Cleaning and preliminary analysis of benchmark web logs, **(b)** Emulated test bed to fabricate attack traffic, **(c)** Constructing training datasets from *EXP* Access logs, **(d)** Performance evaluation of classifier models

Table 1 represents all the attack strategies and their respective parameters value. GET flood attacks are usually generated with an inter-request delay value close to 300 ms (Beitollahi and Deconinck, 2013, 2012; Lu and Yu, 2006). We randomized this value between 100 and 1000, and increased the upper bound for the inter-request delay to make the attacks more sophisticated. The feature value may increase or decrease depending on the employed attack strategy. Table 1 also represents the direction of possible fluctuations in the values of features with respect to the attack strategies. Such changes in one or more features aid the discrimination of bots and legitimate users.

1. Experimental Design

The work conducted in this study is divided into four subsequent phases, as shown in Fig. 4. Our approach relies on formulating baseline behaviors of legitimate users and attacking bots from their respective browsing activities. The browsing activities are computed in terms of the four features. To begin with, we employ four web logs containing access traces of the legitimate users. It may be noted that the complete experimentation shown in Fig. 4 is independently carried out for each web log. The web log is initially investigated for the removal of partial or inappropriate entries and then pre-processed for the extraction of the information required for attack fabrication and feature set computations (Fig. 4a). Using the attack emulation test bed, the access traces of various types of GET flood attack strategies are fabricated (Fig. 4b) and merged with the traffic from legitimate web log to form *EXP* access log. *EXP* access log, containing traces of both legitimate users and attacking bots, is processed to compute the typical ranges of feature set values representing legitimate and attack baseline behaviors, respectively. These feature values are used to construct a training dataset comprising the labeled attack and legitimate instances. The training dataset is then used to build a number of classification models using the machine learning classifier algorithms as shown in Fig. 4c. The performance of these models is finally evaluated in terms of various standard metrics. The following subsections give through information regarding various operations constituting each phase.

* 1. Web log cleaning and preliminary analysis

The data extracted from benchmark web logs form the basis of forthcoming phases. Therefore, it becomes essential to perform cleaning in order to avoid false results arising from the possible inconsistencies present in the web logs. Initially, we remove all the record entries with any missing fields. IP address field usually contains resolved names and addresses, which we substitute with unique 5-digit identifiers (UID) for future processing. Further, the fields (user string, etc.) not relevant to our study are removed, thus reducing the size of web logs. Finally, the web logs are sorted according to the user identifiers (UID) and server access time, which allow us to perform user level processing.

|  |  |
| --- | --- |
|  |  |
| **(a)** WorldCup98 | **(b)** Clarknet |
|  |  |
| **(c)** NASA | **(d)** University |
| **Fig. 5**. Number of overall requests received by the server in time window Tn | |

During the preliminary analysisphase, we compute various prerequisite parameters that are required to compute feature values during subsequent phases. This analysis process is followed for individual web logs separately to build their respective baseline behaviors. Fig. 5 shows the overall (from all users) number of requests received by the server for different web logs. The number of requests (*η*) each user makes to the server in time period Tn is initially calculated to obtain the cumulative percentage frequency distribution as shown in Fig. 6. The range of requests rates, shown in Fig. 6, is divided into four classes using the Chebyshev’s Inequality theorem (Pukelsheim, 1994). This range defines the limits of individual classes of set *Ɍ*. As discussed in Section 3, the class transitions are also determined in this phase to aid the calculation of feature *Ƒ*1. Table 2 shows the number of transitions from initial class *x* to class *y* in different web logs. The response size distribution (*ƥ*) defining the amount of data (bytes) requested by individual users in time window Tn is shown in Fig. 7. The feature *Ƒ*2 takes this distribution as an input to derive its value.

|  |  |
| --- | --- |
|  |  |
| **(a)** WorldCup98 | **(b)** Clarknet |
|  |  |
| **(c)** NASA | **(d)** University |
| **Fig. 6**. Log-lin frequency-request plot | |

* 1. Traffic traces preparation

To the best of our knowledge, there are no benchmark web logs that contain GET flood attack traces. The researchers, as a result, use various simulation and emulation techniques to fabricate the attack traces. We used the hybrid test bed DDoSTB (Behal and Kumar, 2016) that comprises 75 physical nodes, 3 physical routers, and 5 switches. It also contains an 8-core Linux server acting as the attack target. This test bed combines real and emulated traffic generating nodes allowing us to produce real-time attack traffic traces. We use two set of cluster nodes for legitimate users and attack bots respectively. These two clusters *Cluster 1* and *Cluster 2* connects to the server through 2 layer-2 switches, 2 layer-3 switches, 1 firewall, and 3 routers, as shown in Fig. 8.

**Table 2** Number of transitions among classes in *Ɍ* for different datasets.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Initial Class (*x*) | Final Class (*y*) | Transition Counts | | | |
| WorldCup98 | University | NASA | Clarknet |
| Low | Low | 32769 | 1328 | 8142 | 53162 |
| Low | Normal | 15418 | 739 | 3559 | 34753 |
| Low | High | 646 | 49 | 433 | 2215 |
| Low | Spurious | 48 | 1 | 5 | 33 |
| Normal | Low | 14425 | 775 | 4202 | 36574 |
| Normal | Normal | 25014 | 773 | 4205 | 54521 |
| Normal | High | 3784 | 93 | 521 | 4554 |
| Normal | Spurious | 91 | 3 | 9 | 50 |
| High | Low | 1093 | 88 | 481 | 2010 |
| High | Normal | 3101 | 102 | 746 | 5215 |
| High | High | 734 | 4 | 150 | 2922 |
| High | Spurious | 13 | 1 | 4 | 47 |
| Spurious | Low | 42 | 5 | 11 | 42 |
| Spurious | Normal | 93 | 7 | 10 | 62 |
| Spurious | High | 36 | 0 | 1 | 43 |
| Spurious | Spurious | 4 | 0 | 0 | 0 |

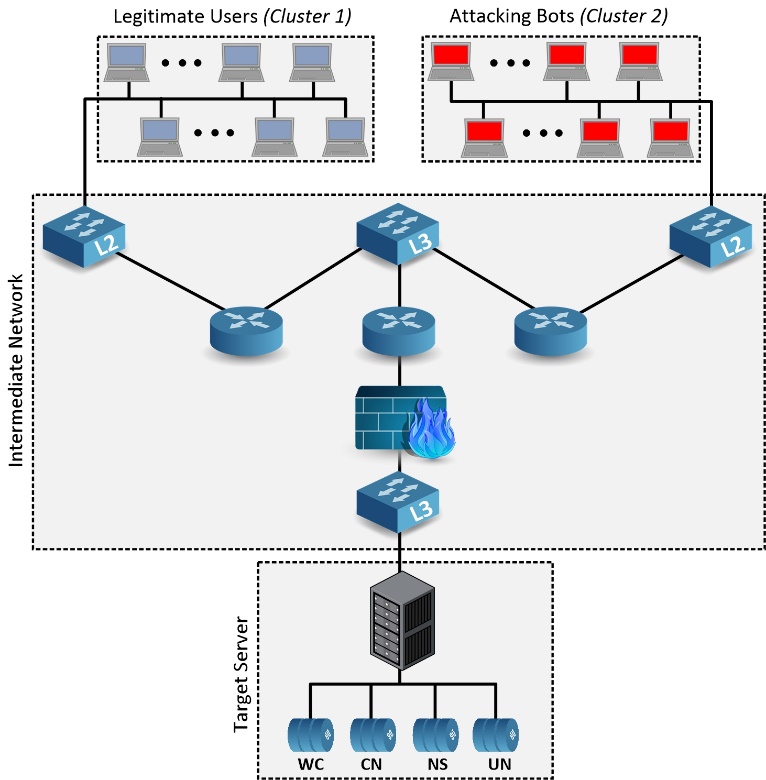
|  |  |
| --- | --- |
|  |  |
| **(a)** WorldCup98 | **(b)** Clarknet |
|  |  |
| **(c)** NASA | **(d)** University |
| **Fig. 7**. Aggregated response data (bytes) requested by the users | |

Legitimate traffic traces of benchmark web logs are replayed from *Cluster 1*. It is worth mentioning that the portions of the benchmark web logs replayed in this phase are different from portions used in the preliminary analysis phase. This is done in order to reduce the possible bias that may occur in validating the feature values for legitimate users. Continuous and sequential streams of attack traffic pertaining to 12 attack strategies, each separated with variable differences (30 sec ≤ δ ≤ 120 sec), are generated using the bot cloud on *Cluster 2*. *Cluster 2* is deployed with multiple instances of Apache JMeter[[2]](#footnote-2) to form a cloud of 3K active attack bots (Rajab *et al.*, 2007). JMeter has been used by some recent studies (Di *et al.*, 2013; Saleh and Abdul Manaf, 2015) associated with GET flood attacks. The availability of multiple customization options (Halili, 2008) allowed us to generate 12 varying GET flood attack strategies. Different pools of IP addresses are assigned to users on *Cluster 1* and *Cluster 2* for facilitating source identification.

In the preliminary analysis, every benchmark web log is individually parsed using a python script to identify resources (web object names and their respective sizes) that were originally stored on the servers when logs were collected. This information is used to build four databases on the server containing dummy files of respective names and sizes associated with each of the benchmark web logs. The target server allows access to these resources from the user and attack clouds on *Cluster 1* and *Cluster 2* respectively. Every request attempt made by legitimate users and attack bots are stored with relevant information in the access logs of the target server. For future reference, these logs as termed as *EXP* access logs.

* 1. Model training

*EXP* access logs collected on the target server are processed to build datasets for training the machine learning classifiers. This processing follows the procedure shown in Fig. 4c. Every user session is initially split into sub-sessions with time length Tw. Features *Ƒ*3 and *Ƒ*4 are computed for these sub-sessions. Further, we split sub-sessions into four segments with time length Tn. The variables computed during the preliminary analysis are used to compute features *Ƒ*1 and *Ƒ*2. Algorithm 1 shows the pseudo-code for computing the feature set values for individual users. Finally, we prepare four training datasets corresponding to four benchmark web logs. The instances in these datasets are labeled either as legitimate users or as attack bots, based on their originating IP addresses.



**Fig. 8.** Emulation test bed for attack fabrication

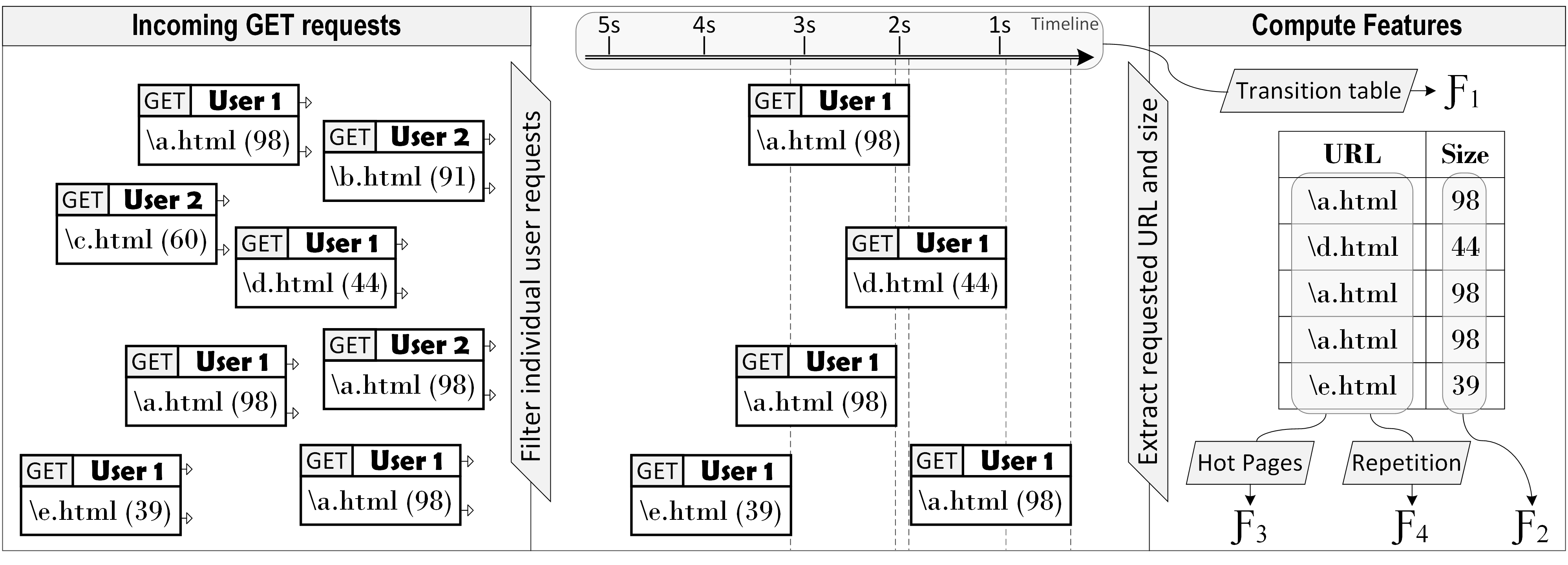
The process of training begins by allowing the machine learning algorithms to build classification models from the training dataset. Every input sample in the training datasets comprises feature set values and the respective class value. The models constructed by the machine learning classification algorithms split the feature space into decision regions equal to the number of input classes. Any new unclassified sample is categorized to a single and most probable class based on the projection of its features values onto the decision regions. The underlying method followed by different machine learning algorithms decides the apportioning of the feature space into decision regions.

|  |  |
| --- | --- |
| **Algorithm 1 Pseudo code to compute feature set values of users.** | |
| **01:** | Initialize *Tn* as 30 seconds and *Tw*as 120 seconds |
| **02:** | Input: Ɍ limits and *ϰ* |
| **03:** | **WHILE** observed sessions are not done **do** |
| **04:** | Split each user session into sub-sessions with time period of *Tw* |
| **05:** | **FOR-ALL** observed sub-sessions **do** |
| **06:** | Split Sub-sessions into four segments each with time period of *Tn* |
| **07:** | **FOR-EACH** segment **do** |
| **08:** | Classify the number of requests based on *Request\_States* limits |
| **09:** | Compute *Ƒ2*based on Eqn. (7) for each sub  window |
| **10:** | **END FOR-EACH** |
| **11:** | Compute transition score for consecutive segments  i.e. *Ƒ1* using Eqn. (3) |
| **12:** | Average *Ƒ1*and *Ƒ2*over four sub windows |
| **13:** | Compute *Ƒ3*and *Ƒ4*based on Eqn. (8)and Eqn. (11), respectively |
| **14:** | **END FOR-ALL** |
| **15:** | **END WHILE** |
|  |  |

In this study, the categorization involves two classes, legitimate and bot. The experiments presented in this study are carried out in Weka (Holmes *et al.*, 1994), which provides a stable GUI to support investigation of a number of machine learning classifiers. Weka supports different validation modes essential to analyze the effectiveness of resultant classification models. The four training datasets prepared are fed to machine learning algorithms in Weka to build classification models.

* 1. Feature calculation and prediction

Four datasets previously prepared were used to construct 24 classification models, each associated with one of the six machine learning classifiers considered by this study. We use k-fold cross-validation mode that is known to provide highly reliable validation results for classification performance. A 10-fold cross-validation is used to evaluate classifier performance in categorizing each user as legitimate or bot. However, we can also evaluate the performance of the classification models using real-time incoming traffic. In order to evaluate the detection performance for real-time traffic, the incoming traffic is processed to extract feature set values. For every individual user, the feature set values are calculated after the completion of time window Tw. *Python* is used to implement the logic behind the extraction of feature values from the incoming user traffic traces. The procedure to extract feature values from the real-time traffic is same as that followed during the training stage. Fig. 9 illustrates the process of computing feature set values of users from the incoming requests.



**Fig. 9.** Feature calculation from incoming real-world traffic

1. Complexity Analysis and Deployment

It is important to analyze the time and space complexities of the proposed detection system. The time and space complexities for preliminary web log analysis are almost O(1), therefore, are not included in the discussion. This subsection evaluates the time and space complexities of our proposed detection system.

* 1. Time complexity

Let us consider the time complexities for computing the features *Ƒ1*, *Ƒ2*, *Ƒ3*,and *Ƒ4* are , and respectively. There are 3 types of operations OP1, OP2, and OP3 performed during computations of the feature values. OP1 comprises all the single memory fetch operations that include functions such as (*x,y*), *ƥ*(t), and *η*(t). We also add operations like addition, deletion, natural log, and exponential, etc in OP1. OP2 performs a few comparisons (less than 4) and a single fetch operation, and include the function *Ɍ*(*n*). The functions in OP3 performs a large number of comparisons before returning. W(E) and P(*i*) are dependent on the number of unique web objects requested by the user in time window Tw. If a user makes request to *WB* unique web objects then the worst case time complexity for these two functions is . The time required to calculate each of the feature individually in the following text.

The time complexity to calculate feature *Ƒ1* is . Similarly, for feature *Ƒ2*, . The time complexity of computing feature values of *Ƒ3*and *Ƒ4* are dependent on the number of web objects that are requested by a user in time window Tw. From the equations we compute and . Therefore, the feature set computation for a single user costs

For users, the total time complexity will be . For simplicity in calculating the time complexity, we consider the operations OP1 and OP2 takes 1 and 3 units of CPU time. The total time complexity therefore becomes . Based on the preliminary analysis of the web logs, the maximum value of *WB* is 4×114 i.e. length of Tw and maximum number of requests made by a user in single Tn (which is 114 for WorldCup98 dataset). Consider there are 100000 active users connected to a web server. Clearly at any time, the number of active users are very large than *WB* i.e. . The time complexity for computing feature set after the end of time window Tw is .

* 1. Space complexity

As the server monitors and stores variables corresponding to each of the connected users, the space complexity therefore becomes vital. The stored variables are used for calculation of the feature values after completion of Tw. The proposed system uses three variables is a sequential array that stores the identifiers associated with web object requested by a user within time window Tw. are the arrays of length 4, which sequentially store the request counts and cumulative response size respectively for the 4 consecutive time windows Tn.

Clarknet dataset comprised of the maximum number of unique web objects i.e. 20103. 15 bits are required to uniquely identify each web object in the dataset. In the worst case, the maximum length of is equal to the maximum number of requests in time window Tw i.e. 4×114 = 456. The total size of becomes 6840 bits. Each element in can be of size 8 bits as the maximum value of request count in a single time window Tn is 114. The total size of becomes 32 bits. The maximum value of upper limit computed after removing outliers is 196155 for WorldCup98 dataset. In the worst case suppose a user keeps on requesting a web object with size of 196155 bytes, the maximum value of cumulative response with 114 requests could reach upto 114×196155 i.e. 22361670 bytes. We need 25 bits in order to store this number in bytes or 15 bits if stored in units of KiloBytes. The total size of becomes 100 bits (taking 25 bits each).

So, in the worst case, we require 6840+32+100 = 6972 bits, less than 1KB, to capture the behavior of a single user during the time window Tw. This memory is re-used to store new values after every Tw, which a modern-day web server can easily handle.

* 1. Deployment

There are two main deployment options for a defense system, standalone at the target server or distributed in the network. The researchers usually prefer deployment at the victim end due to the nature of GET flood attacks requires. In the proposed system, once the model is trained it is installed as a Web Application Firewall (WAF) capable of sniffing the traffic going toward the server. WAF acts as a moderator between a user and the web server. The feature set values for each user are extracted from the traffic and the trained model is used to classify the user as legitimate or attack bot. The IP addresses of classified potential bots are sent to the server for necessary actions.

The legitimate users frequently make use of online web proxies to access a server. Also, the users in universities or other organizations access the online services from behind the NAT. Consequently, the server receives traffic of a large number of users through a small set of IP addresses. These IP addresses due to their high request rates are on the verge of being classified as attack bots. One approach to avoid such scene is to whitelist these IP addresses (web proxies or NATs). However, it is not possible to fully legitimize the traffic from proxies and NATs because an attacker may direct the bot requests through these proxies (if public). GET flood detection mechanisms are usually deployed at the server end. Therefore, it is not possible to individualize the proxy traffic into discrete users. The proposed system scrutinizes individual user behavior instead of monitoring the traffic as a whole, thus, it is practically deployable at web proxies.

If there is a change in the deployment location then the classification model has to be re-calibrated based on the destined web server’s access log. This requires the procedures depicted in Fig. 4a, Fig. 4b, and Fig. 4c to be executed for the respective web log in order to build the desired classification model. The sufficient details of these procedures provided in Section 4 allow readers to reproduce or repeat the same experiment. It is worth mentioning that our work evaluated the performance of various machine learning algorithms using datasets constructed from web logs and generalized one of them as the most efficient (SVM in our case). When conducting the same experiment with any other web log, the classification results may vary as compared to the results reported in this work depending on the nature of user traffic in that respective log. Consider we need to implement this approach for protecting our university website then the web log pertaining to our university website is used to build an exclusive classification model. As a part of validation, this is demonstrated by employing two-day accesses from our university web log in this experiment along with the other three benchmark web logs.

1. Classification Results and Discussion

This section reports the results obtained from the experimental evaluations in three steps. Firstly, we perform a model selection with 10-fold cross-validation in order to select a model that efficiently generalizes the data. Secondly, the classification performance of the proposed system is tested during a flash event (background traffic) using the WorldCup98 web logs. Finally, our proposed system is assessed against previous works on detection of GET flood attacks. The following subsection outlines the evaluation parameters used to compare and examine the performance of different classifier models.

* 1. Performance evaluation parameters

Initially, we compute four primitive metrics namely TN, FP, FN, and TP, which form the basis of many other parameters required for the performance evaluation of classifiers. TN quantifies legitimate user instances in legitimate zone, FP quantifies legitimate user instances in attack zone, FN quantifies bot instances in legitimate zone, and TP quantifies bot instances in attack zone. The values of these metrics are used to compute precision and recall, which further are used to compute F1-Score (Eqn. 13). Detection Rate (Eqn. 14) and False Positive Rate (Eqn. 15) are used to evaluate the performance of the proposed features in detecting attack bots. Relative Root Squared Error (RRSE) is computed using Eqn. (16).

|  |  |
| --- | --- |
|  | (13) |
|  | (14) |
|  | (15) |
|  | (16) |

Matthews correlation coefficient (MCC) and Kappa statistic (KS) are also computed using Eqn. (17) and Eqn. (20). These are considered suitable parameters to quantify the quality of binary classifications even with the unequal sizes of attack and legitimate training instances. The values of area under ROC curve and Precision-Recall Curve (PRC) are also computed.

|  |  |
| --- | --- |
|  | (17) |
|  | (18) |
|  | (19) |
|  | (20) |

In addition to these, two metrics complexity and scalability are used to evaluate the performance of the detection works. Complexity refers to the ease of deployment of a scheme in a real-world environment. High complexity indicates deployment difficulty of the technique in the real world. This is possible if the technique requires the deployment of multiple modules, demands frequent updates, etc. Scalability refers to the applicability of a technique in high workload environment. Low scalability indicates that the underlying technique relies on heavy computations and overheads, therefore is unsuitable for high workload environment.

* 1. Classification results

In the experimental stage of this study, four labeled datasets, each defining one of the four benchmark web logs, are constructed to train and test various machine learning classifiers. The performance of each classifier for different benchmark web logs (Fig. 10) is depicted in the form of Receiver Operating Curve (ROC), which determines the trade-off between TP rate and FP rate. ROC curves representing the performance of each classifier indicate low false positive rate and high precision or true positive rate. The maximum detection rate of 98.33% is achieved by SVM classifier model for NASA dataset. NASA web log corresponds to a mid-level server i.e. the average number of requests received by these servers are considerably less than the other three (WorldCup98, Clarknet, and University). As a result, the high rate attacks are easily visible to the detection system.

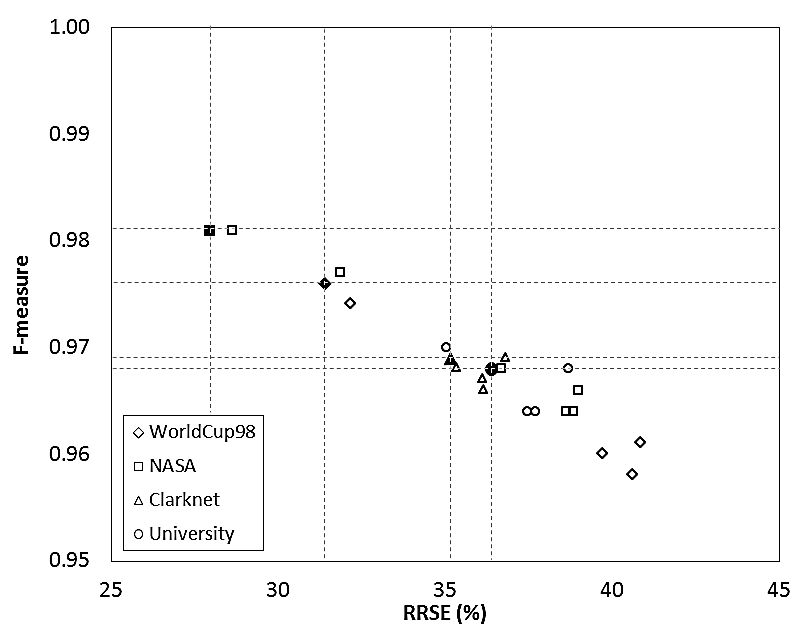
|  |  |
| --- | --- |
|  |  |
| **(a)** WorldCup98 | **(b)** Clarknet |
|  |  |
| **(c)** NASA | **(d)** University |
| **Fig. 10.** Comparison of ROC curves for different datasets | | |

The accesses from proxy and NAT users in benchmark web logs WorldCup98 and Clarknet web logs were not removed during the data cleaning process. The sharing of single or a pool of IP addresses by multiple users might have affected the baseline behavior of features *Ƒ1* and *Ƒ2*. The classifiers' performance is not much affected as the classification model takes into consideration all four features including features *Ƒ3* and *Ƒ4*. Table 3 shows the results of different classifier models using performance evaluation parameters.

Fig. 11 depicts the F1-score and respective RRSE values corresponding to various classifier models evaluated against four datasets. We highlight four classifier models, one from each dataset, having maximum F1-score and minimum RRSE values. Evidently, SVM classifier establishes reasonable performance across the four datasets.

**Table 3** Performance evaluation of different classifier models

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Classifiers | DR (%) | FPR | F1-score | MCC | ROC Area | PRC | KS | RRSE (%) |
| WorldCup98 | | | | | | | | |
| Naïve Bayes | 94.54 | 0.061 | 0.945 | 0.887 | 0.966 | 0.957 | 0.8871 | 45.2863 |
| Random Forest | 96.08 | 0.04 | 0.961 | 0.92 | 0.962 | 0.949 | 0.9196 | 40.8736 |
| SVM | 97.58 | 0.028 | 0.976 | 0.951 | 0.974 | 0.964 | 0.9506 | 31.3917 |
| IBK | 92.08 | 0.088 | 0.921 | 0.836 | 0.916 | 0.889 | 0.8359 | 57.135 |
| JRIP | 95.79 | 0.043 | 0.958 | 0.914 | 0.955 | 0.944 | 0.9137 | 40.6391 |
| J48 | 95.99 | 0.041 | 0.96 | 0.918 | 0.956 | 0.942 | 0.9177 | 39.7278 |
| Clarknet | | | | | | | | |
| Naïve Bayes | 89.74 | 0.104 | 0.897 | 0.793 | 0.958 | 0.948 | 0.7932 | 52.5322 |
| Random Forest | 96.89 | 0.033 | 0.969 | 0.937 | 0.969 | 0.96 | 0.9372 | 36.8433 |
| SVM | 96.93 | 0.033 | 0.969 | 0.938 | 0.968 | 0.955 | 0.9381 | 35.154 |
| IBK | 94.47 | 0.056 | 0.945 | 0.889 | 0.947 | 0.928 | 0.8886 | 47.1796 |
| JRIP | 96.79 | 0.035 | 0.968 | 0.936 | 0.96 | 0.948 | 0.9352 | 35.352 |
| J48 | 96.70 | 0.036 | 0.967 | 0.934 | 0.962 | 0.955 | 0.9334 | 36.1555 |
| NASA | | | | | | | | |
| Naïve Bayes | 96.44 | 0.06 | 0.964 | 0.922 | 0.97 | 0.972 | 0.9204 | 38.6372 |
| Random Forest | 96.44 | 0.055 | 0.964 | 0.922 | 0.965 | 0.963 | 0.9208 | 38.8488 |
| SVM | 98.33 | 0.031 | 0.981 | 0.958 | 0.975 | 0.971 | 0.9578 | 27.9366 |
| IBK | 96.61 | 0.041 | 0.966 | 0.924 | 0.964 | 0.955 | 0.924 | 38.9939 |
| JRIP | 97.68 | 0.037 | 0.977 | 0.948 | 0.968 | 0.963 | 0.9476 | 31.8602 |
| J48 | 96.84 | 0.054 | 0.968 | 0.931 | 0.954 | 0.949 | 0.9294 | 36.694 |
| University | | | | | | | | |
| Naïve Bayes | 90.69 | 0.09 | 0.907 | 0.811 | 0.961 | 0.95 | 0.8102 | 48.7775 |
| Random Forest | 96.76 | 0.038 | 0.968 | 0.933 | 0.973 | 0.967 | 0.9332 | 36.3754 |
| SVM | 97.01 | 0.038 | 0.970 | 0.939 | 0.966 | 0.955 | 0.9382 | 35.0481 |
| IBK | 94.17 | 0.062 | 0.942 | 0.88 | 0.943 | 0.92 | 0.8801 | 48.9151 |
| JRIP | 96.41 | 0.043 | 0.964 | 0.926 | 0.956 | 0.944 | 0.9259 | 37.454 |
| J48 | 96.16 | 0.072 | 0.968 | 0.922 | 0.958 | 0.947 | 0.9206 | 38.673 |

****

**Fig. 11.** Model selection using F-measure and RRSE values

* 1. Discriminating legitimate FE users from attack bots

We evaluate the proposed approach in discriminating the attack bots from the legitimate users during a flash event. Flash event (FE) refers to a situation when a large number of legitimate users simultaneously requests a server (Loukas *et al.*, 2007). This event usually occurs during or after a major event capable of capturing the interest of an enormous number of users. As a result, the server is clogged due to the request flood generated by these users. The overloaded server may get the wrong impression of the existence of an on-going attack during a flash event. Fig. 12 shows a high overlap of the range of feature set values attained by the legitimate users during normal days and flash event.

We prepared a dataset using the experimental setup explained in Subsection 4.2 to train SVM classifier using 10-minute traffic trace on the 66th day of WorldCup98 web log. SVM classifier is chosen based on the results discussed in the previous subsection. The detection accuracy of 99.1% with RRSE 19.2% is achieved using SVM classifier. If an attacker tries to launch GET flood attack while an on-going FE, the proposed detection system can effectively identify attack bots from the legitimate user base. Consequently, the proposed system is able to differentiate between GET flood DDoS attack and FE based on the scores of the users.

|  |  |
| --- | --- |
|  |  |
| **(a)** Distribution of Ƒ1 values | **(b)** Distribution of Ƒ2 values |
|  |  |
| **(c)** Distribution of Ƒ3 values | **(d)** Distribution of Ƒ4 values |
| **Fig. 12**. Feature set values of legitimate users during normal background traffic and flash event. | |

* 1. Comparison with existing works

Various studies in the existing literature have targeted the detection of various GET flood attack strategies. A few of them have only focused on recognizing the presence of these attacks. However, a mere detection would limit the applicability of the detection system in the real world. In order to completely eradicate the impact of these attacks on the server, the primary aim of the detection should be the identification of individual spurious elements (bots). Our detection system operates on users individually to characterize them as legitimate or bot based on their past behavior.

Table 4 compares the detection performance of the proposed system with the existing works that have utilized the corresponding benchmark web logs. Table 5 presents an overall comparison of the proposed technique with the related work using five performance parameters. The values of DR and FPR corresponding to related works are calculated by taking an average over DR values and FPR values across their respective investigated datasets and attack strategies. The number of attack strategies studied by our research work and other previous works, mentioned in Table 5, is determined based on the taxonomy defined in our previous survey (Singh *et al.*, 2017b).

**Table 4** Comparison with the existing works for different benchmark web logs

|  |  |  |
| --- | --- | --- |
| Works | DR | FPR |
| WorldCup98 | | |
| Huang *et al.* (2014) | 97% | 2% |
| Xie and Yu (2009b) | 90% | 1% |
| Clarknet | | |
| Liao *et al.* (2014) | 99.80% | 0.44% |
| Liao *et al.* (2015) | 89.25% | 3.5% |
| NASA | | |
| Suen *et al.* (2010) | 90% | 30% |
| Proposed work | 97.46% | 3.25% |

**Table 5** Comparison of the proposed system with existing works

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Works | DR | FPR | Complexity | Scalability | Number of attack strategies |
| Huang *et al.* (2014) | 97% | 2% | Medium | Low | 2 |
| Liao *et al.* (2014) | 99.80% | 0.44% | Medium | High | 3 |
| Liao *et al.* (2015) | 89.25% | 3.5% | Low | High | 4 |
| Suen *et al.* (2010) | 90% | 30% | Low | High | 1 |
| Wang *et al.* (2014) | 88.95% | 5.10% | Low | High | 2 |
| Xie and Yu (2009b) | 90% | 1% | High | Low | 1 |
| Xu *et al.* (2014) | 96% | 3.40% | Low | High | 1 |
| Yadav and Subramanian (2016) | 98.99% | 1.27% | Low | High | 2 |
| Proposed work | 97.46% | 3.25% | Low | High | 12 |

In contrast to the existing literature, our work provides the capability of identifying 12 GET flood attack strategies along with maintaining high detection rate (97.46% using SVM). However, the value of FPR rises due to the inclusion of some high sophisticated GET flood attack strategies. Nonetheless, in order to make sure that a legitimate user is not blocked from accessing the server, the detection system needs to be supplemented by graphical puzzles (CAPTCHA) based detection mechanisms. Our system provides a transparent approach to the attack whereas the CAPTCHAs ensure that the legitimate users that are classified as bots by our system get another chance of proving their legitimacy.

The complexity of the proposed system is low as the trained model can be deployed as a single module with the server. In addition, the system can swiftly be updated due to low learning time of the SVM classifier (even for large datasets). The scalability of the proposed system in the real world is high due to time complexity and very low memory overhead for each user. The increase in the user base of a server will not have a significant impact on the performance of the detection approach.

1. Conclusions and Future Work

This research work focuses on building a machine learning based detection system that utilizes four proposed features to identify GET flood attack strategies by distinguishing bots from the legitimate users. These features take advantage of bot-specific browsing behaviors to capture spurious clients impersonating as legitimate users. The publicly available web logs such as WorldCup98, Clarknet, and NASA along with our University traffic traces are used to prepare attack traffic traces on an emulated test bed. A selected set of machine learning classification algorithms is used to build models that are able to effectively capture bot sources. Among various machine learning classifiers used, SVM achieved a detection rate of 97.46% across all the data sets; thereby outperforming other classifiers. Moreover, the efficiency of the model built using SVM for the detection of bots during a flash event (extracted from WorldCup98 web log) attained a detection rate of 99.1%. The proposed system supports the detection of 12 different attack strategies with suitably high detection rate and scalability by limiting the computational and space complexity. Instead of only indicating the presence of attacks, our system pinpoints the malicious sources to assist the filtering process by the network administrators. The major contributions and findings of this research work are summarized as follows:

1. Clarknet web log contains a high volume of traffic forwarded by web proxies. Consequently, the detection rate is the lowest and false positives are the highest in the case of Clarknet web log. Henceforth, the overall performance values of our system were affected by low detection rate values for Clarknet web log.
2. There are no benchmark web logs that contain the traces of GET flood attacks. As a result, researchers practice fabricating the traffic traces and validating them with legitimate benchmark traffic traces. For legitimate traffic traces, we used three web logs WorldCup98, Clarknet and NASA from the year 1998 (except our university data set recorded in the year 2016). Based on our literature review (Singh *et al.*, 2017b), these 3 web logs have been widely utilized for experimenting in the studies associated with detection of GET flood DDoS attacks.
3. The average request frequency of a user in time window *Tn* in WorldCup98, University, and NASA is very less as evident from their respective *Low* to *Low* transition counts with values of 32769, 1328, and 8142, respectively. However, the average request frequency in Clarknet web log is considerably high as evident from its *Normal* to *Normal* transition count value of 54521. WorldCup98 and Clarknet are considered as large-sized web servers. Clarknet web log constituted a maximum number of unique web objects i.e. 20103. On the other hand, the web servers pertaining to NASA and University web logs are considered small-sized because of their very few unique web objects 1817 and 618, respectively. Attributing to these differences, similar attack strategies may produce dissimilar impact across different web servers. As a result, the proposed system needs site-specific calibration prior to its deployment. The historical web logs of the server, where defense mechanism is to be deployed, are used to extract relevant information (popular pages, request frequency, and response size distributions, etc.), and to train the detection model.
4. We performed a rank analysis of the four proposed features using *Ranker* search method in Weka tool. The features were ranked as *Ƒ1*,*Ƒ2*, *Ƒ3,* and *Ƒ4* based on the scores associated with their information gain. The features *Ƒ1*and*Ƒ2*offer significantly high detection efficiency for a few number of GET flood attack strategies. However, with the introduction of other sophisticated attack strategies, the features *Ƒ3* and *Ƒ4*highly contribute toward the overall performance of the proposed detection system.
5. The feature values for each client are computed at the end of every wide time window *Tw*i.e. after every 120 seconds. Thus, the delay in detection of the bots for an ongoing attack will also be 120 seconds. As the wide windows of 120 seconds are composed of four 30-second windows, lowering the duration of narrow time windows significantly reduces the detection performance. Apparently, the proposed system with larger time windows is more efficient in terms of detection accuracy. The duration of time windows taken in this study has been finalized based on the trade-off between the desired detection delay and accuracy.
6. Nowadays, the concept of super-botnet is gaining popularity. Super-botnets are identified by their massive number of active bots. In the absence of a super-botnet i.e., if the sufficient number condition holds, the proposed features *Ƒ1*, *Ƒ2,*and *Ƒ3* are able to provide decent detection accuracy. The detection accuracy is likely to degrade when an attacker is able to accumulate bots (super-botnet) beyond the legitimate user base. This is because an attacker could lower the request rates to evade many frequency-based detection techniques. In this case, the feature *Ƒ3* becomes more functional as it exploits the concept of popularity for the identification of bots. In the worst case, if an attacker is able to maintain the super-botnet and accumulate popular web pages, then it can mimic a GET flood attack that will almost be impossible for many of the state-of-the-art detection techniques to differentiate from the normal traffic.
7. Many legitimate users make use of online web proxies to access the server. Also, the users in universities or other organizations access the Internet from behind the NAT. As a result, the server receives traffic of a large number of users through a small set of IP addresses. Therefore, it is not possible to individualize the proxy traffic from different users. As a result, these IP addresses are on the verge of being identified as attack bots in case of heavy traffic conditions. It is not possible to fully legitimize the traffic from proxies and NAT as an attacker may instruct its bots to request through these proxies (if public). GET flood detection mechanisms are usually deployed at the server end. However, the proposed approach can be implemented in a distributed manner at proxy systems, as it functions by explicitly inspecting traffic from individual users.

This research opens up a number of avenues for future work. Some of them are as follows:

1. The detection rate of the proposed system is likely to degrade if the number of active bots supersedes the number of legitimate users connected to the server. Our future work will focus on providing effective detection even in the presence of such super-botnets. We will also investigate the possible solutions to increase the detection rate while maintaining other performance parameters in a suitable range.
2. Future work will include extending the proposed system to a filtering framework to provide a comprehensive protection solution against HTTP-GET flood attacks.
3. The research work is constrained to fabricate the attack traffic traces for GET flood attacks because of the non-availability of desired benchmark web logs. Although there are daily instances of GET flood attacks, the reluctance of the victim organizations to publicize their web logs is the primary reason for such a scenario. This research work can be extended if there is an availability of any public GET flood attack web log in the future.
4. The proposed system can be installed on the web proxies to protect from attacks that use these proxies as intermediate destinations. The efforts toward further reducing the complexity of the proposed scheme can be made so as to provide a light-weight solution that can easily be deployed on web proxies and/or NATs.
5. Our detection system pre-requisites calibration of multiple parameters based on the historical web logs of the server to be protected. For deployment on different server locations, the calibration process is repeated to build different models. Future research could include providing a standardized solution that does not demand site-specific calibrations. Then again, it is challenging due to the variations of user volumes across high-end and low-end servers.

Acknowledgements

This Research work is supported by the All India Council for Technical Education (AICTE), New Delhi, India under Research Promotion Scheme (RPS) under Grant No. 8023/RID/RPS-93/201112.

References

Bad bot landscape report, 2016. https://resources.distilnetworks.com/white-paper-reports/2015-bad-bot-landscape-report, Bot Report. *Distil Networks*.

Behal, S. and Kumar, K., 2017. Detection of DDoS attacks and flash events using novel information theory metrics. *Computer Networks*, vol. 116, pp. 96–110. DOI: 10.1016/j.comnet.2017.02.015

Behal, S. and Kumar, K., 2016. Measuring the impact of DDoS attacks on web services - A realtime experimentation. *International Journal of Computer Science and Information Security*, vol. 14, pp. 323–330.

Beitollahi, H. and Deconinck, G., 2013. ConnectionScore: A statistical technique to resist application-layer DDoS attacks. *Journal of Ambient Intelligence and Humanized Computing*, vol. 5, pp. 425–442. DOI: 10.1007/s12652-013-0196-5

Beitollahi, H. and Deconinck, G., 2012. Tackling application-layer DDoS attacks. *Procedia Computer Science*, pp. 432–441. DOI: 10.1016/j.procs.2012.06.056

Bhandari, A., Sangal, A.L., Kumar, K., 2016. Characterizing flash events and distributed denial-of-service attacks: An empirical investigation. *Security and Communication Networks*, vol. 9, pp. 2222–2239. DOI: 10.1002/sec.1472

Bhatia, S., Schmidt, D., Mohay, G., Tickle, A., 2014. A framework for generating realistic traffic for distributed denial-of-service attacks and flash events. *Computers and Security*, vol. 40, pp. 95 – 107. DOI: 10.1016/j.cose.2013.11.005

Chang, C.-C., Lin, C.-J., 2011. LIBSVM: A library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27. DOI: 10.1145/1961189.1961199

Chen, S., Tang, Y., & Du, W., 2007. Stateful DDoS attacks and targeted filtering. *Journal of Network and Computer Applications*, vol. *30*, 823–840. DOI: 10.1016/j.jnca.2005.07.007

Choi, Y.S., Oh, J.T., Jang, J.S., Kim, I.K., 2011. Timeslot monitoring model for application layer DDoS attack detection, In: *Proceedings of the International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, pp. 677–679.

Cui, Y., Yan, L., Li, S., Xing, H., Pan, W., Zhu, J., and Zheng, X., 2016. SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks. *Journal of Network and Computer Applications*, vol. *68*, pp. 65–79. DOI: 10.1016/j.jnca.2016.04.005

Di, X.Q., Yang, H.M., Qi, H., 2013. Low-Rate application-Layer DDoS attacks detection by Principal Component Analysis (PCA) through user browsing behavior. *Applied Mechanics and Materials*, vol. 397, pp. 1945–1948. DOI: 10.4028/www.scientific.net/AMM.397-400.1945

Du, P. and Nakao, A., 2010. OverCourt: DDoS mitigation through credit-based traffic segregation and path migration. *Computer Communications*, vol. 33, pp. 2164–2175. DOI: 10.1016/j.comcom.2010.09.009

Giralte, L.C., Conde, C., de Diego, I.M., Cabello, E., 2013. Detecting denial of service by modelling web-server behaviour. *Computers and Electrical Engineering*, vol. 39, pp. 2252–2262. DOI: 10.1016/j.compeleceng.2012.07.004

Halili, E.H., 2008. Apache JMeter: A practical beginner’s guide to automated testing and performance measurement for your websites. *Packt Publishing Ltd*.

Holmes, G., Donkin, A., Witten, I.H., 1994. WEKA: A machine learning workbench, In: *Proceedings of the Second Australian and New Zealand Conference on Intelligent Information Systems*, pp. 357–361. DOI: 10.1109/ANZIIS.1994.396988

Huang, C., Wang, J., Wu, G., Chen, J., 2014. Mining web user behaviors to detect application layer DDoS attacks. *Journal of Software*, vol. 9. DOI: 10.4304/jsw.9.4.985-990

Hubert, M. and Vandervieren, E., 2008. An adjusted boxplot for skewed distributions. *Computational Statistics and Data Analysis*, vol. 52, pp. 5186–5201. DOI: 10.1016/j.csda.2007.11.008

Jazi, H.H., Gonzalez, H., Stakhanova, N., Ghorbani, A.A., 2017. Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling. *Computer Networks*, vol. 121, pp. 25–36. DOI: 10.1016/j.comnet.2017.03.018

Kshirsagar, D. and Kumar, S., 2016. HTTP flood attack detection using ontology, In: *Proceedings of the International Conference on Advances in Information Communication Technology and Computing*, USA, pp. 15:1–15:4. DOI: 10.1145/2979779.2979794

Lee, S., Kim, G., Kim, S., 2011. Sequence-order-independent network profiling for detecting application layer DDoS attacks. *EURASIP Journal on Wireless Communications and Networking*, vol. 2011:50. DOI: 10.1186/1687-1499-2011-50

Liao, Q., Li, H., Kang, S., Liu, C., 2015. Application layer DDoS attack detection using cluster with label based on sparse vector decomposition and rhythm matching. *Security and Communication Networks*, vol. 8, pp. 3111–3120. DOI: 10.1002/sec.1236

Liao, Q., Li, H., Kang, S., Liu, C., 2014. Feature extraction and construction of application layer DDoS attack based on user behavior, In: *Proceedings of the 33rd Chinese Control Conference*, pp. 5492–5497. DOI: 10.1109/ChiCC.2014.6895878

Loukas, G., Öke, G., Kalita, J.K., 2007. A denial of service detector based on maximum likelihood detection and the random neural network. *Computer Journal-Oxford*, vol. 50, pp. 717–727.

Lu, W. z, Yu, S. z, 2006. An HTTP flooding detection method based on browser behavior, In: *Proceedings of the International Conference on Computational Intelligence and Security*, pp. 1151–1154. DOI: 10.1109/ICCIAS.2006.295444

Miu, T., Wang, C., Luo, D.X., Wang, J., 2016. Modeling user browsing activity for application layer DDoS attack detection, In: *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pp. 747–750. DOI: 10.1007/978-3-319-59608-2\_42

Ni, T., Gu, X., Wang, H., Li, Y., 2013. Real-Time detection of application-layer DDoS attack using time series analysis. *Journal of Control Science and Engineering*, vol. 2013, article ID e821315. DOI: 10.1155/2013/821315

Pukelsheim, F., 1994. The three sigma rule. The American Statistician 48, pp. 88–91.

Rajab, M.A., Zarfoss, J., Monrose, F., Terzis, A., 2007. My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging, In: *Proceedings of the First Conference on Hot Topics in Understanding Botnets*, USA, pp. 1–8.

Saleh, M.A., Abdul Manaf, A., 2015. A novel protective framework for defeating HTTP-based denial of service and distributed denial of service attacks. *The Scientific World Journal*, vol. 2015, article ID e238230. DOI: 10.1155/2015/238230

Singh, K., Singh, P., Kumar, K., 2017a. Impact analysis of application layer DDoS attacks: A simulation study. *International Journal of Intelligent Engineering Informatics*, vol. 5, pp. 80–100. DOI: 10.1504/IJIEI.2017.082564

Singh, K., Singh, P., Kumar, K., 2017b. Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges. *Computers and Security*, vol. 65, pp. 344–372. DOI: 10.1016/j.cose.2016.10.005

Singh, K., Singh, P., Kumar, K., 2016. A systematic review of IP traceback schemes for denial of service attacks. *Computers and Security*, vol. 56, pp. 111–139. DOI: 10.1016/j.cose.2015.06.007

Sree, T.R. and Bhanu, S.M.S., 2016. HADM: Detection of HTTP GET flooding attacks by using analytical hierarchical process and Dempster–Shafer theory with MapReduce. *Security and Communication Networks*, vol. 9, pp. 4341–4357. DOI: 10.1002/sec.1611

Suen, H.Y., Lau, W.C., Yue, O., 2010. Detecting anomalous web browsing via diffusion wavelets, In: *Proceedings of the IEEE International Conference on Communications*. pp. 1–6. DOI: 10.1109/ICC.2010.5502089

Wang, J., Yang, X., Long, K., 2011. Web DDoS detection schemes based on measuring user’s access behavior with large deviation, In: *Proceedings of the IEEE Global Telecommunications Conference*, pp. 1–5. DOI: 10.1109/GLOCOM.2011.6133798

Wang, J., Yang, X., Zhang, M., Long, K., Xu, J., 2014. HTTP-SoLDiER: An HTTP-flooding attack detection scheme with the large deviation principle. *Science China Information Sciences*, vol. 57, pp. 1–15. DOI: 10.1007/s11432-013-5015-2

Wang, X., Abraham, A., & Smith, K. A., 2005. Intelligent web traffic mining and analysis. *Journal of Network and Computer Applications* *28*, pp. 147–165. DOI: 10.1016/j.jnca.2004.01.006

Worldwide Infrastructure Security Report (No. XII), 2017. https://pages.arbornetworks.com/rs/082-KNA-087/images/12th\_Worldwide\_Infrastr ucture\_Security\_Report.pdf, *Arbor Networks*.

Xiao, P., Qu, W., Qi, H., Li, Z., 2015. Detecting DDoS attacks against data center with correlation analysis. *Computer Communications*, vol. 67, pp. 66–74. DOI: 10.1016/j.comcom.2015.06.012

Xie, Y., Tang, S., Xiang, Y., Hu, J., 2013. Resisting web proxy-based HTTP attacks by temporal and spatial locality behavior. *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 1401–1410. DOI: 10.1109/TPDS.2012.232

Xie, Y. and Yu, S., 2009a. A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM Transactions on Networking*, vol. 17, pp. 54–65. DOI: 10.1109/TNET.2008.923716

Xie, Y. and Yu, S.-Z., 2009b. Monitoring the application-layer DDoS attacks for popular websites. *IEEE/ACM Transactions on Networking*, vol. 17, pp. 15–25. DOI: 10.1109/TNET.2008.925628

Xu, C., Zhao, G., Xie, G., Yu, S., 2014. Detection on application layer DDoS using random walk model, In: *Proceedings of the IEEE International Conference on Communications (ICC)*, pp. 707–712. DOI: 10.1109/ICC.2014.6883402

Xuan, Y., Shin, I., Thai, M.T., Znati, T., 2010. Detecting application denial-of-service attacks: A group-testing-based approach. *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 1203–1216. DOI: 10.1109/TPDS.2009.147

Yadav, S. and Subramanian, S., 2016. 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 (ICCTICT)*, pp. 361–366. DOI: 10.1109/ICCTICT.2016.7514608

Ye, C. and Zheng, K., 2011. Detection of application layer distributed denial of service, In: *Proceedings of the International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 310–314. DOI: 10.1109/ICCSNT.2011.6181964

Ye, C., Zheng, K., She, C., 2012. Application layer DDoS detection using clustering analysis, In: *Proceedings of the International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 1038–1041. DOI: 10.1109/ICCSNT.2012.6526103

Yu, J., Fang, C., Lu, L., Li, Z., 2010. Mitigating application layer distributed denial of service attacks via effective trust management. *IET Communications*, vol. 4, pp. 1952–1962.

Yu, S., Guo, S., Stojmenovic, I., 2015. Fool me if you can: Mimicking attacks and anti-attacks in cyberspace. *IEEE Transactions on Computers*, vol. 64, pp. 139–151. DOI: 10.1109/TC.2013.191

Yu, S., Zhou, W., Jia, W., Guo, S., Xiang, Y., Tang, F., 2012. Discriminating DDoS attacks from flash crowds using flow correlation coefficient. *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, pp. 1073–1080. DOI: 10.1109/TPDS.2011.262

Zhang, H., Taha, A., Trapero, R., Luna, J., Suri, N., 2016. SENTRY: A novel approach for mitigating application layer DDoS threats, In: *Proceedings of the IEEE Trustcom and BigDataSE and ISPA*, pp. 465–472. DOI: 10.1109/TrustCom.2016.0098

Zhou, Wei, Jia, W., Wen, S., Xiang, Y., Zhou, Wanlei, 2014. Detection and defense of application-layer DDoS attacks in backbone web traffic. *Future Generation Computer Systems*, vol. 38, pp. 36–46. DOI: 10.1016/j.future.2013.08.002

1. http://www.nitj.ac.in [↑](#footnote-ref-1)
2. http://jmeter.apache.org/ [↑](#footnote-ref-2)