Journal of Information Security and Applications 65 (2022) 103090

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

HTTP request pattern based signatures for early application layer DDoS detection: A firewall agnostic approach

Amit Praseed [*], P. Santhi Thilagam

Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, India

ARTICLE INFO

Keywords:
Application layer
DDoS
Early detection
HTTP request patterns
Dynamic signatures

ABSTRACT

Application Layer DDoS (AL-DDoS) attacks are an extremely dangerous variety of DDoS attacks that started becoming popular recently. They are executed using very few legitimate requests, making them very difficult to detect. Since they are executed using attack generation tools and botnets, AL-DDoS attacks display similarity within a request stream (temporal similarity) and across request streams (spatial similarity). Once a particular request stream has been detected as malicious by an anomaly detection mechanism (ADM), spatial similarity can help in detecting AL-DDoS attacks much earlier by employing a dynamic signature based approach. In this work, we use HTTP request patterns as signatures to build a firewall agnostic Early Detection Module (EDM) for AL-DDoS attacks. We also propose the use of Sample Entropy instead of the popular Shannon's Entropy to identify AL-DDoS attacks. Sample Entropy is able to model both the frequencies and sequence of data items within a request stream, and is a better indicator of temporal similarity than Shannon's Entropy. In this work, we demonstrate that Sample Entropy can be used effectively to detect AL-DDoS attacks. With a Sample Entropy based anomaly detection mechanism, we demonstrate that the use of EDM significantly reduces the detection latency for AL-DDoS attacks.

1. Introduction

Distributed Denial of Service (DDoS) attacks are some of the most dangerous attacks against web applications. Traditionally, DDoS attacks have been a staple at the network layer, but in recent years, these attacks have started penetrating the application layer as well. Attacks at the application layer are significantly more complicated than their counterparts at the network layer, and possess certain features which make them difficult to detect. Application Layer DDoS (AL-DDoS) attacks can be executed with a very low bandwidth which can easily bypass existing volume based detection mechanisms. In addition, they are strikingly similar to legitimate user traffic, which further complicates detection. In recent years, a more sophisticated variation of these attacks have been observed, which use computationally expensive requests to launch attacks. These attacks, called asymmetric attacks, have a very low bandwidth and cannot be detected by any of the existing firewalls [1].

DDoS attacks are often executed using a large network of compromised systems, called botnets. The Mirai botnet, which surfaced in 2016, has since been held responsible for a number of high profile DDoS attacks. It has been shown to be able to leverage both network and application layer protocols to launch DDoS attacks. Mirai launched 15,194 attacks between September 2016 and February 2017, out of which 35% were at the application layer [2]. In August 2017, a new botnet named WireX was discovered attempting to disturb various servers and CDNs using a large volume of HTTP GET requests. It was through the concerted efforts of researchers from Akamai, Cloudflare, Flashpoint, Google, Oracle Dyn, RiskIQ, Team Cymru, and other organizations that this botnet was identified and combated without much incident [3].

A closer inspection of the source code of DDoS attack generation tools and botnets reveal that DDoS attacks often tend to follow a predefined script or plan of attack, which lends an inherently repetitive nature to DDoS traffic [4]. This means that DDoS attack traffic displays a considerable amount of similarity. Even though more advanced versions of attack generation tools and botnets possess capabilities to randomize attack traffic using multiple user agents and query strings, they form only a small percentage of attacks. A considerable proportion of DDoS attacks still use non-randomized attacks [5–8] due to the ease with which they can be used owing to the lack of a reconnaissance step.

In particular, we define two kinds of similarity: temporal and spatial. Temporal similarity means that a particular pattern repeats periodically within a single attacking connection while spatial similarity refers to the fact that the same pattern repeats across multiple attacking connections. The presence of temporal similarity in a request sequence can be used to identify if a particular connection is malicious. On the

[*] Corresponding author.
E-mail address: amitpraseed@gmail.com (A. Praseed).

https://doi.org/10.1016/j.jisa.2021.103090

Available online 19 January 2022

other hand, spatial similarity in request traffic can be exploited to enable early detection of DDoS attacks by employing a dynamic signature based approach. Dynamic signatures are widely used in firewalls to combine the flexibility of anomaly detection techniques with the speed of signature detection techniques [9–11]. Such an approach employs an anomaly detection technique to detect previously unknown attacks, and stores a signature corresponding to the attack in a dictionary. Subsequent occurrences of the attack can be detected by referring to the signature dictionary, and need not be processed by the anomaly detection unit.

The repeating pattern or signature present in DDoS attack traffic varies depending on the type of attack. In network layer DDoS attacks, this repeating pattern is often a particular IP header field value or the request inter-arrival duration [11]. However, these signatures cannot be used for detecting application layer DDoS attacks without incurring a significant number of false positives. This is because attacks at the application layer are carried out using legitimate requests, and it is not possible to detect these attacks by searching for patterns within requests. The most effective way to detect these attacks is to examine the request sequence generated by a client [12].

Application Layer DDoS attacks are generated by scripts or attack generation tools where the attacker specifies a set of URLs in the target web application. The attacking tool then iterates over the set of URLs continuously for a predefined duration. This exact same behaviour is followed by all the bots that are executing the attack. This mode of attack means that application layer DDoS attacks consist of a set of URLs that keep repeating periodically [13], which can be used as an attack signature. Certain botnets such as the Blackenergy botnet send repeated requests to only a single URL [14] to constitute a single URL flood. Botnets such as DirtJumper and Yoddos have provisions to target multiple URLs to constitute a multiple URL flood [15,16]. In either case, the use of the repeating request sequence or pattern as the attack signature can lead to more efficient detection of application layer DDoS attacks. Such an approach allows the system to detect malicious clients early and reduces the amount of redundant computation performed for anomaly detection.

In this work, we propose the use of HTTP request patterns as attack signatures for the early detection of application layer DDoS attacks. The proposed Early Detection Module (EDM) can be used in combination with any firewall with AL-DDoS detection capabilities in order to reduce the attack detection latency of the firewall. The EDM works independently of the Anomaly Detection Module (ADM) in the firewall, and only provides some input to the ADM which can be used to detect DDoS attacks early. In this sense, the proposed EDM is firewall agnostic. The EDM works by extracting the repeating URL sequence from an attack stream to form an attack signature. The use of this attack signature can help detect subsequent occurrences of the attack comparatively early. To showcase the efficiency of the EDM, we combine the EDM with two different AL-DDoS detection mechanisms — the first of which is a new detection approach based on Sample Entropy while the second is an automata based detection mechanism from our previous work [17] — and show that the use of the EDM results in significantly reduced attack detection latency in both cases. The AL-DDoS detection mechanisms are fundamentally different, which underlines the fact that the proposed EDM is firewall agnostic.

While demonstrating the efficiency of the EDM, this work also introduces a new approach for detecting application layer DDoS attacks by exploiting the temporal similarity in a request stream. Earlier attempts at detecting DDoS attacks by observing temporal similarity used Shannon's Entropy to describe the randomness in a request sequence. Entropy of a data stream only depends on the frequency of elements in a data stream, and not their sequence. In the context of DDoS attack detection, however, the presence of a repeating sequence of requests is a crucial factor, and cannot be modelled effectively by entropy [18]. A modified version of entropy, which takes into account the sequence of elements in addition to their frequency is Sample Entropy (SampEn) and can serve as a much better indicator of temporal similarity in DDoS traffic. As part of this work, we demonstrate that Sample Entropy can be used to detect application layer DDoS attacks. More importantly, we demonstrate how the EDM speeds up the detection of application layer DDoS attacks.

Our work is different from the existing research works in two aspects. Firstly, to the best of our knowledge, none of the existing research works have used request patterns as attack signatures in a dynamic signature based approach to detect application layer DDoS attacks. Secondly, we demonstrate that Sample Entropy can be used effectively to detect application layer DDoS attacks. To the best of our knowledge, no research work has been done on the utility of Sample Entropy in detecting application layer DDoS attacks.

Our contributions in this work are as follows:

- We present a dynamic signature based detection mechanism based on HTTP request patterns to detect non-randomized application layer DDoS attacks early.
- The proposed mechanism is firewall agnostic, and can be integrated easily with any existing web application firewall.
- We demonstrate that Sample Entropy (SampEn) can be used effectively to detect Application Layer DDoS attacks, and proves to be a better indicator of DDoS attacks than regular entropy.

2. Terminology

In this section, we formally define the concepts that were introduced in the previous section in the context of application layer DDoS attacks. Consider an application layer DDoS attack generated by a single client, consisting of a sequence of HTTP requests $R = \{r_1, r_2 ... r_i, \dots, r_j ...\}$. Each request r_i has a number of attributes (such as method, URL, user agent etc.), which are represented as $r_i.A$, where A is the name of the attribute.

Definition 2.1. Temporal Similarity means that during a non-randomized DDoS attack, $\exists A, k \ such\ that$
$\forall r_i, r_i.A = r_{i+k}.A$
where k is called the Repeat Length and is denoted by Repeat Length(R), and A is the Repeat Attribute denoted by RepeatAttrib(R).

The part of R represented by $r_i.A, r_{i+1}.A, \dots, r_{i+k-1}.A$ is called the Repeating Pattern.

In the case of an AL-DDoS attack, this Repeat Attribute is often the URL. In a DDoS attack, there are often thousands of such attacking clients $R_1, R_2...$ and it is possible to identify similarities between these clients as well.

Definition 2.2. Spatial Similarity means that during a non-randomized DDoS attack, $\forall i, j$,
$RepeatAttrib(R_i) = RepeatAttrib(R_j)$
$RepeatLength(R_i) = RepeatLength(R_j)$

As discussed earlier, spatial similarity between DDoS attack clients can be exploited to enable the early detection of AL-DDoS attacks. Consider an attack client with request sequence $R = \{r_1, r_2 ... r_i, \dots, r_j ...\}$. Let us represent the DDoS detection mechanism as a function $D()$ that inspects the request sequence as it comes, and return $True$ if an attack is detected. In other words, $D()$ successively evaluates $D(r_1 ... r_i)$ till it returns $True$ or the sequence terminates. The decision length (DL), n, for a DDoS attack stream R with respect to a detection mechanism D is the smallest value of i for which $D(r_1 ... r_i)$ returns True.

Definition 2.3. An auxiliary mechanism E is said to enable Early DDoS Detection for a DDoS detection mechanism D if the use of E in combination with D creates a lower decision length than when D is used alone, i.e., $DL(E \circ D(R)) < DL(D(R))$

3. Related work

3.1. Similarity in DDoS traffic

DDoS attacks are generated using attack generation tools coupled with a network of compromised systems called a botnet. The perpetrators of the attack often do not participate directly in the attack. Rather, they control compromised systems (bots) through a central Command and Control (C&C) server. The bots periodically communicate with the C&C server to receive instructions regarding the target and the mode of attack. Since they operate based on a specific set of instructions, botnet traffic exhibits a high degree of similarity or repetition. A significant amount of work as been done on identifying botnets by observing C&C traffic, which is the periodic communication between a bot and its C&C server [19–21]. However, bot detection by observing C&C traffic is complicated due to the need for large scale network monitoring.

A high degree of similarity exists in DDoS attack traffic executed using botnets as well. This allows for server side botnet traffic detection, which is another viable option. Network layer DDoS attacks often have repeating flow patterns and header fields which allow for their efficient detection. The situation is more difficult in the case of application layer DDoS attacks, where the use of request rates or header fields as measures of similarity often result in a significant number of false positives. Research by Bukac and Matyas [13] has showed that monitoring the request streams for repeating URL patterns can be one of the most important techniques for detecting HTTP based DDoS attacks at the server side. This is further corroborated by the fact that request sequence often forms the backbone for the efficient detection of application layer DDoS attacks.

In particular, botnet based DDoS traffic displays two distinct types of similarity — spatial and temporal. Spatial similarity in DDoS traffic refers to the fact that attack traffic coming from different, possibly geographically separated, bots display significant similarity. Temporal similarity in DDoS traffic means that traffic coming from a single bot is itself composed of similar units repeating periodically. This similarity could be in terms of repeating URL patterns, header fields, request parameters etc. This similarity in DDoS traffic allows it to be detected with a fair degree of accuracy.

3.2. Early DDoS detection using spatial similarity

DDoS attack detection is fundamentally an intrusion detection problem. A large number of network layer DDoS attacks can be detected using signature based intrusion detection techniques because they carry specific signatures within their packets. This is not the case with application layer DDoS attacks, which are executed using legitimate requests, and contain no specific signatures within the requests that can potentially aid in their detection. Anomaly detection techniques are commonly used to identify such attacks. In particular, anomaly detection techniques using request sequences work very well in detecting even sophisticated application layer DDoS attacks. Anomaly detection techniques are behaviour based and can detect previously unknown attacks. However, a drawback of anomaly detection techniques is that they perform a lot of redundant computation and are hence slower in comparison to signature based approaches.

A large amount of work has been done concerning early detection of network layer DDoS attacks. Some of these approaches use more sophisticated features which leads to early detection [22], while a few others enable early detection by setting suitable threshold values for attack detection [23]. However, majority of the early DDoS detection approaches use a collaborative approach [24–28]. A collaborative approach works by pooling the knowledge acquired by multiple network layer devices and end-users. By the time a network layer DDoS attack reaches an end-user, it has assumed massive proportions, and becomes extremely difficult to mitigate. A better strategy is to identify comparatively smaller surges in network traffic at intermediate devices, and combine that knowledge to decide whether a DDoS attack could be impending. A collaborative approach allows for the excess traffic to be curtailed at the intermediate routers so that the DDoS attack does not reach its full potential.

Dynamic Signature based techniques capitalize on Spatial Similarity in DDoS attacks to accelerate the detection process. The system consists of an anomaly detection module which monitors request traffic and identifies previously unknown attacks. Once an attack is detected, an attack signature is generated for the attack and stored in a dynamic signature dictionary. Subsequent incidences of the attack can be detected using a signature based approach, which avoids the anomaly detection process and reduces detection time.

Dynamic signature based techniques essentially rely on pattern matching algorithms to detect the presence of malicious signatures in data streams. In the context of anomaly detection, majority of the data stream will be benign. Checking the pattern dictionary for every single user access adds a non-negligible execution cost which cannot be justified. It makes more sense to check the pattern dictionary only in the case of suspected malicious patterns. The use of Bloom Filters can help in this regard. A Bloom Filter is a space-efficient probabilistic data structure that is used to test set membership. The basic idea behind a Bloom Filter is that every element is hashed with a number of hash functions, and the corresponding positions in the Bloom Filter is set. To check if an element is present in a Bloom Filter, the element is hashed with the same set of hash functions, and the corresponding bits are checked. If all the bits are set, the element could be present in the set. If all the bits are not set, then the element is definitely not present in the set.

When Bloom Filters are used in network layer signature detection, malicious signatures are added to a dictionary as well as a bloom filter [29–32]. When an incoming packet needs to be checked for the presence of a malicious pattern, it is first checked with the bloom filter. Bloom filters can confirm the non-inclusion of a pattern with complete accuracy in constant time. Thus, the use of bloom filters avoids the need of checking the pattern dictionary for benign patterns. However, bloom filters cannot be used as a replacement for searching the pattern dictionary because bloom filters can generate false positives. So, when the bloom filter returns positive for the presence of a malicious pattern, it must be confirmed using an actual pattern search before making a decision. However, since most of the incoming traffic is likely to be benign, the use of bloom filters significantly improves the running time of the pattern search.

3.3. Detecting application layer DDoS attacks using anomaly detection

Detection of application layer DDoS attacks is primarily focused on identifying anomalies in the request stream statistics [12]. The features used in this analysis include request rate, request inter-arrival duration, session inter-arrival duration, percentage of valid requests etc, entropy of the request stream etc. Of these, request stream entropy has been used extensively to great effect. Entropy measures the information content within a data stream. In the DDoS context, entropy of a request stream gives an idea about the randomness (or order) in a request stream. The rationale behind using request entropy for the detection of application layer DDoS attacks is that legitimate users often request a wider variety of resources than attack request streams. Since they are generated using automated tools, attack request streams tend to be repetitive in nature. In other words, entropy tries to measure the temporal similarity of an attacking connection.

Entropy has been widely used in the detection of network layer DDoS attacks [33–37]. In the context of application layer DDoS attacks, the use of entropy is often limited to observing the entropy within request statistics. Zhou et al. [38] used model entropy but their work was meant for identifying attack streams in backbone web traffic. Zhao et al. [39] used two measures — Entropy of URL per IP (EUPI) and Entropy of IP per URL (EIPU) for identifying flooding attacks.

During a random flooding attack or a fixed URL flooding attack, EUPI would increase, thus indicating an attack. EIPU helped in distinguishing attacks and flash crowds. Singh et al. [40] used the entropy of GET requests per connection and the variation of the entropy per IP address to detect flooding attacks.

These detection mechanisms are able to detect volumetric application layer DDoS attacks but do not generalize well to sophisticated attacks because they do not use request sequence as the basis for computing entropy. The use of entropy to observe the randomness in request sequence has been comparatively rare. Devi and Yogesh [41] used the concept of request stream entropy along with trust values for users to detect attacks. Jin Wang et al. [42] used relative entropy to compare the observed click ratio of users to that of legitimate users in order to detect application layer DDoS attacks.

When used to measure temporal similarity within a URL sequence, the traditional entropy measure proposed by Shannon does not work well. This is because Shannon's entropy measures the similarity without considering the sequence of occurrence. This repetition of request sequences is the basis of DDoS traffic using botnets, and needs to be captured using a different measure. Sample Entropy (SampEn) provides a means of measuring temporal similarity while considering the sequence of occurrence of symbols as well [18].

4. Early DDoS detection using request patterns as signatures

In this section, we present the detailed working of our Early DDoS detection module (EDM). The EDM is an add-on module that can be used along with any Anomaly based Application Layer DDoS detection module (ADM) and enables early detection of attacks by using request patterns as dynamic signatures. In line with our definition of Early Detection as per Definition 2.3, our EDM is the auxiliary function E, while the ADM is the AL-DDoS detection mechanism D. It is important to note that our EDM is completely independent of the ADM in use, and hence can be used in conjunction with any ADM.

4.1. Overview

Our Early Detection Module (EDM) is designed to exploit the spatial similarity that exists within an AL-DDoS attack in order to enable early detection. The Repeat Attribute in this case is the attack URL, which could either be a single URL or a periodic repetition of multiple URLs. Consider an AL-DDoS attack executed by k attack clients $C_1, C_2 \ldots C_k$ sending a sequence of HTTP requests to the client. Let us denote the sequence of requests sent by each client as $R_1, R_2 \ldots R_k$. However, in a non-randomized AL-DDoS attack, we can represent all of these request sequences using a single sequence $R = \{r_1, r_2, r_3, r_1, r_2, r_3 \ldots\}$. Let the server be equipped with an ADM (which runs an anomaly detection algorithm D), and an EDM (which runs an algorithm E). The system operates in three phases, which are explained below.

1. Regular Detection: When the system starts, the EDM has no information regarding the temporal or spatial similarity within the attack stream, and hence cannot participate in the detection process. During this phase, the ADM uses its algorithm D to continuously evaluate the incoming sequences of requests to identify whether they are malicious or not. At some point after the attack starts, the ADM detects that the request sequence R_i originating from one of the clients C_i is malicious after n requests, i.e. $DecisionLength(D(R_i)) = n$.
2. Signature Generation: The ADM updates the EDM that R_i is a malicious request sequence, and the EDM generates a signature $S = f(R_i)$ to denote the request sequence R_i. Now the EDM is ready to operate.

Algorithm 1 Algorithm for Identifying Repeating Patterns

Input: Incoming Request Sequence, R
Output: Basic Repeating Pattern, pat

```
$n \leftarrow Length(R)$
$len \leftarrow 0$
$lps[0] \leftarrow 0$
$i \leftarrow 1$
while $i < M$ do
  if $R[i] == R[len]$ then
    $len \leftarrow len + 1$
    $lps[i] \leftarrow len$
    $i \leftarrow i + 1$
  else
    if $len \neq 0$ then
      $len \leftarrow lps[len - 1]$
    else
      $lps[i] \leftarrow 0$
      $i \leftarrow i + 1$
    end if
  end if
end while
$l \leftarrow lps[n - 1]$
if $l > 0$ AND $n\%(n - l) == 0$ then
  $pat = R[0 : (n - l - 1)]$
else
  $pat = R$
end if
return $pat$
```

3. Early Detection: Note that only a single client C_i has been identified as malicious by the ADM. For every other connection, the ADM has to repeat the same process of attack detection, which has to wait for n requests to identify the attack. This is where Early Detection becomes an advantage. Once the initial attack is detected and an attack signature is generated, subsequent instances of the DDoS attack can be detected by using the EDM (aided by the signature S) in addition to the original detection mechanism D. This leads to a reduced Decision Length, thus facilitating early detection. .

Our EDM generates a signature by extracting the smallest repeating request pattern from the attack sequence and storing it in a Malicious Pattern Dictionary. Every subsequent request sequence is inspected for the presence of the malicious patterns in the dictionary. If a malicious pattern is detected, the EDM returns a value to the ADM, which the ADM can use to either block the stream or use it in conjunction with an ongoing anomaly score value. In this way, the EDM simply provides an added input to the ADM, and does not depend on how the ADM processes the requests internally, which makes our approach firewall agnostic. However, searching a dictionary for every request is time consuming, especially since majority of the request sequences will be benign. In order to improve the time consumption, the EDM uses an array of Bloom Filters as an intermediary dictionary to identify benign patterns in constant time. The actual pattern dictionary is searched only when the bloom filter returns positive.

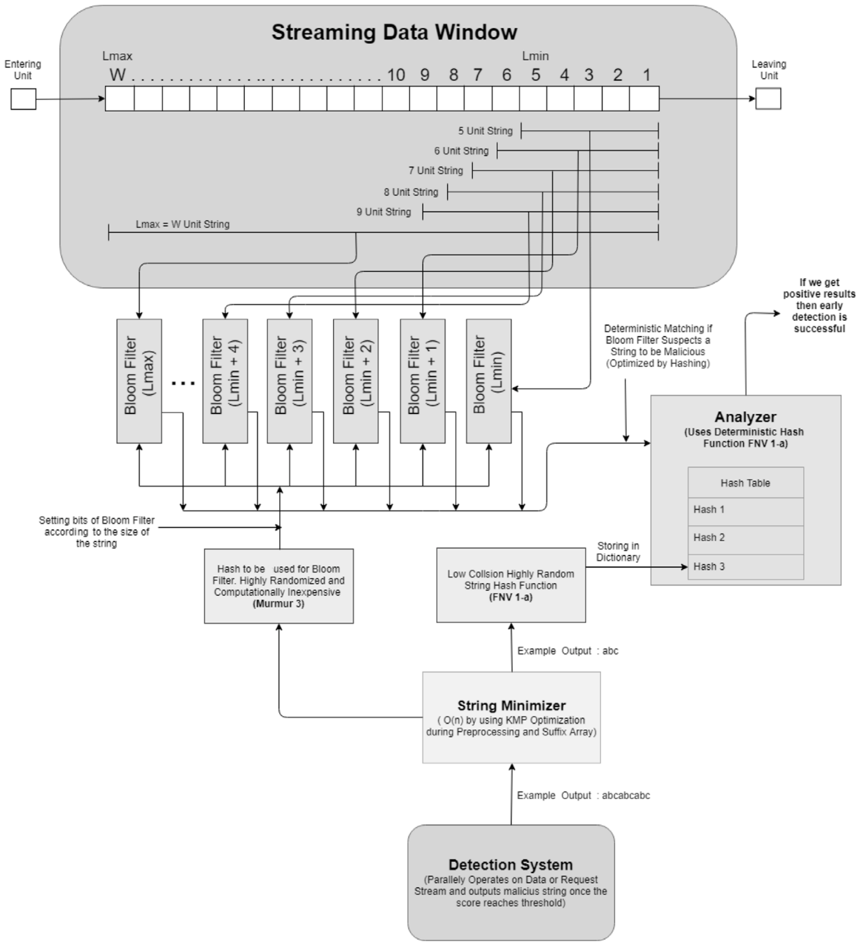
4.2. Architecture

Fig. 1 presents the architecture of our Early Detection Module (EDM). The system is composed of the following components:

- String Minimization Module: Once the ADM identifies a particular connection to be malicious, it passes on the request sequence of

Table 1
Descriptions of symbols used in Algorithm 1.

Symbol	Description
R	Incoming Request Sequence
pat	Repeating request pattern in R (if no pattern is found, then pat will be equal to R)
Length()	Function to compute the length of a string
lps[]	Auxiliary array to store information about the Longest Proper Prefix which is Suffix



Streaming Data Window
Entering Unit
Lmax
W 10 9 8 7 6 5 4 3 2 1
Lmin
Leaving Unit
5 Unit String
6 Unit String
7 Unit String
8 Unit String
9 Unit String
Lmax = W Unit String
Bloom Filter (Lmax)
Bloom Filter (Lmin + 4)
Bloom Filter (Lmin + 3)
Bloom Filter (Lmin + 2)
Bloom Filter (Lmin + 1)
Bloom Filter (Lmin)
Deterministic Matching if Bloom Filter Suspects a String to be Malicious (Optimized by Hashing)
If we get positive results then early detection is successful
Analyzer (Uses Deterministic Hash Function FNV 1-a)
Hash Table
Hash 1
Hash 2
Hash 3
Setting bits of Bloom Filter according to the size of the string
Hash to be used for Bloom Filter. Highly Randomized and Computationally Inexpensive (Murmur 3)
Low Collision Highly Random String Hash Function (FNV 1-a)
Storing in Dictionary
Example Output : abc
String Minimizer (O(n) by using KMP Optimization during Preprocessing and Suffix Array)
Example Output : abcabcabc
Detection System (Parallely Operates on Data or Request Stream and outputs malicius string once the score reaches threshold)

Fig. 1. Architecture of EDM module.

that connection to the EDM. The string minimization module is responsible for extracting the basic repeating unit (or request pattern) from the request sequence. This can be done by any string minimization algorithm. In our implementation, a modification of the Knuth–Morris–Pratt (KMP) algorithm was used for string minimization. The preprocessing step of the KMP algorithm can be used to find the length of the longest proper prefix of the request sequence which is also a suffix. Assuming that the request sequence length is n and the identified substring length is l, the substring contains a repeating pattern if $(n-l)$ divides n. The basic repeating unit is the first $n-l$ characters of the request sequence. Algorithm 1 describes the process of identifying the repeating pattern in a request sequence and Table 1 provides a description of the symbols used in Algorithm 1. After the request sequence is reduced to its basic repeating pattern, it is stored in the Bloom Filter Array as well as the Malicious Pattern Dictionary.

- Malicious Pattern Dictionary (MPD): The Malicious Pattern Dictionary (MPD) stores the hashed values of the malicious request patterns detected by the ADM. The use of hashing prevents the size of the dictionary from growing beyond limits. The primary requirement for the hashing algorithm used is that it should be highly collision resistant as collisions may lead to the blocking of legitimate users. In our implementation, we have used FNV1-a as the hashing algorithm due to its collision resistance.
- Bloom Filter Array (BFA): Searching the pattern dictionary is expensive, hence there is a need to identify positively benign patterns in constant time. The inclusion of an array of bloom filters satisfies this requirement. A Bloom Filter can be used to check the non-inclusion of an element in a set with complete confidence, but it cannot predict the inclusion of an element with certainty due to the problem of false positives. In order to verify the existence of the element in a set after checking with the Bloom Filter, it is necessary to perform deterministic verification of the same. The EDM maintains a separate bloom filter for different lengths of the malicious request patterns. Thus the maximum number of bloom filters used will be $L_{max} - L_{min} + 1$ where L_{max} and L_{min} are the maximum and minimum lengths of the malicious request patterns encountered by the system. Since every request sequence passes through the bloom filters, the hash function used in the bloom filter needs to be computationally inexpensive. In our implementation, we use Murmur3 hash function as it is computationally inexpensive and provides high randomization

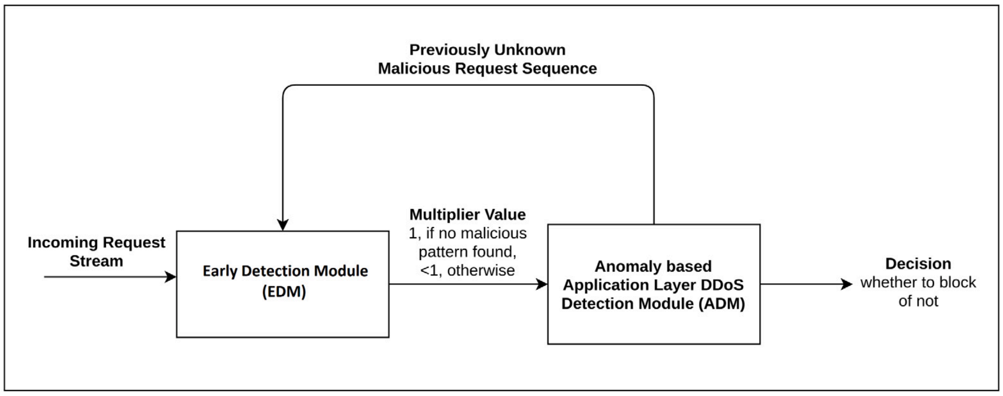

Previously Unknown
Malicious Request Sequence
Incoming Request Stream
Early Detection Module (EDM)
Multiplier Value
1, if no malicious pattern found,
<1, otherwise
Anomaly based Application Layer DDoS Detection Module (ADM)
Decision
whether to block of not

Fig. 2. Working of EDM module.

for preventing false positives. The use of modified Bloom Filters such as Cuckoo Filters [43] can help enable deletions from the dictionary. This allows the size of the dictionary to be maintained within limits.

4.3. Working

The EDM consists of two workflows that operate in parallel. The first workflow is concerned with adding malicious patterns to the MPD. This workflow is initiated by the ADM whenever it detects a previously unknown attack. The ADM forwards the malicious request sequence detected to the EDM. The string minimization module extracts the repeating pattern from the request sequence and stores it in the BFA and the malicious pattern dictionary. This ends the first workflow.

The second workflow operates continuously by inspecting the stream of requests for malicious patterns. A windowing mechanism is employed to extract patterns from the incoming request stream, and the extracted substring is checked for being malicious. The first stage of checking is performed using the BFA. If the BFA confirms that the substring is not malicious, then the examination is abandoned and the substring is deemed to be benign. However, if the BFA sends a positive response, the substring is checked for inclusion in the MPD using a search algorithm. If the substring is present in the MPD, then the substring is malicious. In this case, the EDM sends a value to the ADM indicating that the request stream contains a malicious signature. The ADM can now choose to block the connection immediately, although this approach could lead to false positives. A better approach would be for the ADM to continue with anomaly detection while maintaining an anomaly score based on the value sent. The value of the anomaly score can be manipulated based on the input from the EDM to block malicious connections faster. This allows the EDM to function independently of the mechanism used in the ADM, making it independent of the firewall. A simplified diagram illustrating how the EDM works with an ADM is shown in Fig. 2. It must be noted that the EDM only provides for early detection of AL-DDoS attacks, and does not affect the performance metrics (such as accuracy, false positive rate etc.) of the detection mechanism.

5. Detection mechanisms for AL-DDoS attacks

Considerable research work has gone into the detection of AL-DDoS attacks. In a very broad sense, these detection mechanisms can be broadly classified into those that rely on request statistics and those that rely on request semantics. Detection mechanisms that rely on the statistical features of a request stream usually take into consideration features like request rate, request inter-arrival duration, request stream entropy etc. Mechanisms that rely on request semantics attempt to identify the probability of the actual request sequence encountered. This is usually done by building a model of user behaviour using Markov models or Finite State Machines. In this section we describe two detection approaches for AL-DDoS attacks. In Section 5.1, we describe how the mathematical concept of Sample Entropy can be used to detect AL-DDoS attacks. In Section 5.2, we provide a brief description of one of the already existing detection mechanisms for AL-DDoS attacks. This detection mechanism [17] uses a Probabilistic Timed Automata (PTA) to model user behaviour, and uses this model to effectively detect AL-DDoS attacks.

5.1. Using sample entropy for detecting AL-DDoS attacks

5.1.1. Sample entropy: Describing the regularity of a data stream

Shannon's Entropy, or simple Entropy, is a statistical tool to describe the average rate at which a stochastic source produces information. The information content in a stream of data is defined as the number of bits needed to encode that data stream. For example, a data stream like *aaaaaaa...* which contains repeating entries of a single character contains no information and requires 0 bits to encode. The value of entropy for such a stream is hence, zero. On the other hand, a stream like *abcdef...* which contains all characters different from each other contains a large amount of information.

Shannon's Entropy is defined as:

$$S = -\sum_{i} P_i * log_2(P_i) \tag{1}$$

where the summation is over all possible characters or symbols in the data stream. P_i denotes the probability of occurrence of symbol i in the data stream.

From this definition, a drawback of Shannon's Entropy comes to light. Consider two data streams $A = abcabcabc$ and $B = aabccbba$. According to the definition, both streams A and B have the same value of Shannon's Entropy. However, it is evident that stream A is much more regular than stream. Pincus [18] proposed the used of a new measure of regularity of a data stream in 1991, called Approximate Entropy (ApEn). ApEn is formulated as follows. Given a time series data $u(1), u(2)...u(N)$, the algorithm divides the data into groups of m, i.e. it creates a set of vectors $x(1), x(2)...x(N-m+1)$ such that $x(i) = u(i), u(i+1), \dots, u(i+m-1)$. The algorithm then defines

$$C_i^m(r) = \frac{Number\ of\ \ x(j)\ such\ that\ dist(x(i), x(j)) \leq r}{N-m+1} \tag{2}$$

$C_i^m(r)$ basically defines the number of similar windows, where the window size is m. The algorithm further defines a more general measure of this similarity across all $x(i)$.

$$\Phi^m(r) = \frac{\sum_{i=1}^{N-m+1} logC_i^m(r)}{N-m+1} \tag{3}$$

Since there might be similarities between data windows in even random data, the calculation of ApEn tries to determine whether the similarity in data windows exists even for a larger window size. If the similarity

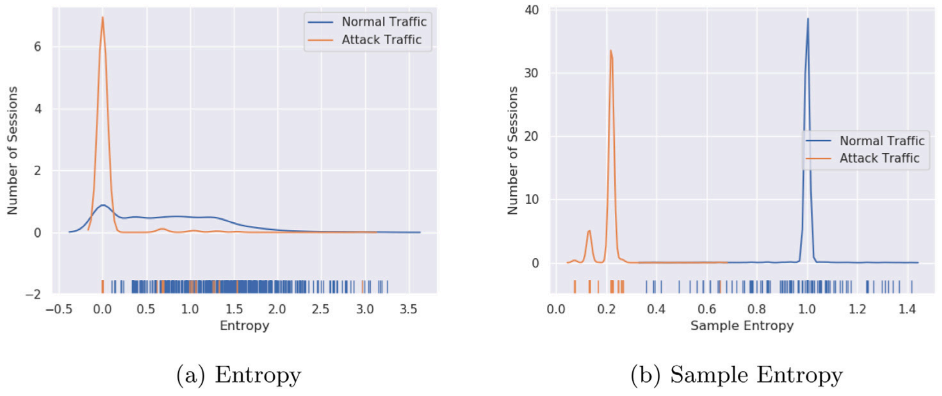


Fig. 3. Comparison of Entropy and Sample Entropy for Detecting AL-DDoS Attacks.

still exists in a larger windows, then the data sequence is more likely to be repetitive.

$$ApEn = \Phi^{m}(r) - \Phi^{m+1}(r) \quad (4)$$

Approximate entropy detects the presence of fluctuating patterns in the input data. A low value of ApEn points to regularity in the data sequence and a higher value indicates randomness. Sample Entropy (SampEn) was introduced as a modification to ApEn. SampEn was meant to remove the drawbacks of ApEn which were dependence on sequence length and redundant calculations.

$$SampEn = log\frac{A}{B} \quad (5)$$

where A is the number of vector pairs having $dist(X_{m+1}(i), X_{m+1}(i)) < r$ and B is the number of vector pairs having $dist(X_m(i), X_m(i)) < r$.

Both Sample Entropy and Approximate Entropy are used extensively in biomedicine to identify anomalies in life cycle processes such as heart rates [44–47].

5.1.2. The validity of sample entropy in detecting attacks

Shannon's Entropy has been used extensively for detecting application layer DDoS attacks alongside statistical features of the request stream such as request rate and inter-arrival duration. When the request sequence is used as the feature, the use of Shannon's Entropy does not fare well. Fig. 3 shows a comparison between Shannon's Entropy and Sample Entropy for normal application layer traffic and DDoS attack traffic. Fig. 3(a) shows that it is difficult to set a proper threshold for Shannon's Entropy which separates normal and attack traffic because of considerable overlap between the values. This is clearly observable in the rugplot at the bottom of Fig. 3(a). However, when using Sample Entropy, the overlap is considerably reduced and it is easy to determine a threshold value which can be used to distinguish between normal and attack traffic with minimal errors. This is clearly observable in Fig. 3(b) and demonstrates that Sample Entropy clearly outperforms Shannon's Entropy in detecting application layer DDoS attacks.

The issue with using SampEn (or Shannon's Entropy) is deciding the proper window size. While the system will often give best results while considering the entire data stream together, this is often not possible in a real time situation with streaming data. In such a scenario, a window size must be selected and SampEn is computed for that window. When new data comes in, the window is slid over and the same process is repeated. A running measure of average SampEn is maintained based on the windows observed so far. Proper selection of window size is crucial for the success of the application.

Fig. 4 shows that SampEn does not aid in detecting attacks when the window size is set too small. This can be observed from the rugplot in Figs. 4(a) and 4(b), where there is considerable overlap between normal and attack SampEn values which makes it difficult to set a threshold. However, as the window size increases, the overlap decreases, and SampEn can be used effectively for attack detection, as can be seen in Fig. 4(h). However, using window sizes that are too large is discouraged because of the increased time and space complexity. A trade-off must be made between efficiency and complexity in selecting the window size. In this work, we have chosen a window size of 45.

5.2. Using probabilistic automata for detecting AL-DDoS attacks

In our earlier work [17], we demonstrated the modelling of user behaviour using a Probabilistic Timed Automata (PTA), along with a suspicion scoring mechanism for detecting AL-DDoS attacks.

The browsing behaviour of users can be effectively modelled using a PTA. In particular, legitimate users tend to abide by certain rules, which prove effective in the detection of AL-DDoS attacks.

- Users tend to browse certain pages and/or page sequences more commonly than others
- Users tend to spend time inspecting the results of a request before issuing a new request
- Users tend to send a mixture of high workload and low workload requests and usually do not send long sequences of high workload requests

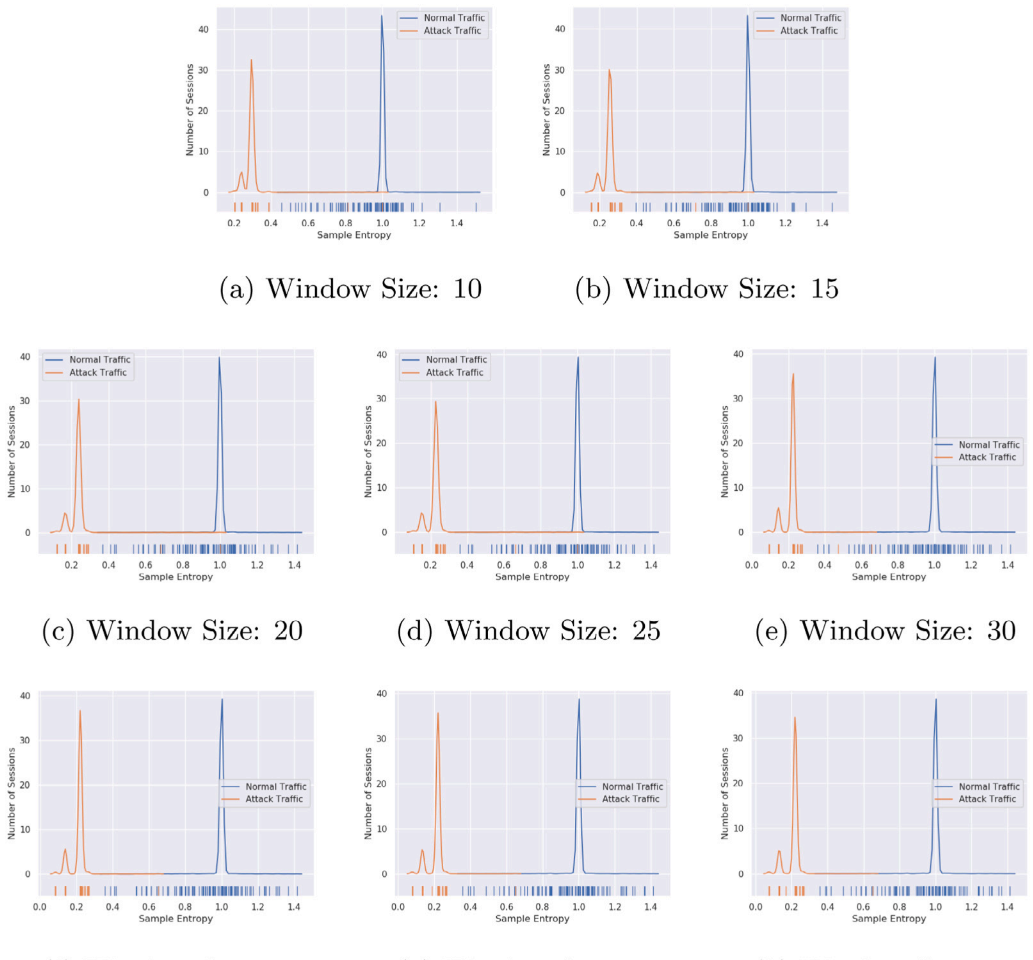
In order to model these three features, in our earlier work [17], we use a PTA with the states representing URLs and the transitions representing requests between URLs. The PTA has three key annotations to model user behaviour. Every transition has an associated transition probability, which denotes the probability with which users make that transition. Every transition also has a think time value which denotes the amount of time that users typically spend in the previous state before making that transition. Every state also has a priority value, which denotes the average server workload of that state. This user behaviour model is constructed from server logs, and the system assigns a suspicion score to incoming connections based on the perceived deviation from this learned model. If the suspicion score exceeds a particular threshold, the connection is blocked.

6. Early attack detection using EDM

In this section, we demonstrate how the EDM can be used to accelerate the detection of application layer DDoS attacks. We also demonstrate the firewall agnostic nature of the EDM by combining the EDM with two fundamentally different ADMs with minimal modification.

6.1. Combining a SampEn based ADM with EDM

The ADM based on SampEn works in two phases: learning and detection. In the learning phase, the ADM receives the system logs as input. The ADM computes the average SampEn for user sessions for the decided window size. Based on the observed values of SampEn,



(a) Window Size: 10 (b) Window Size: 15

(c) Window Size: 20 (d) Window Size: 25 (e) Window Size: 30

(f) Window Size: 35 (g) Window Size: 40 (h) Window Size: 45

Fig. 4. Average Sample Entropy values for different Window Sizes.

a lower threshold value is fixed. This threshold value represents the lowest SampEn value of a stream for which it is considered benign. Any stream with SampEn value below the threshold is classified as malicious in the detection phase.

When the ADM detects any request stream to be malicious, it forwards the request stream to the EDM. The EDM extracts the repeating pattern of the stream as an attack signature and stores it. The EDM also continuously monitors the incoming request stream for the presence of any of the patterns stored in the malicious pattern dictionary. If the EDM detects the presence of a malicious pattern in the incoming request stream, it does not block the user immediately, as doing so would increase the false positive rate. Instead, it passes a multiplier (<1) which is used to reduce the observed sample entropy value for the request stream, so that it crosses the threshold faster. If the occurrence of the malicious pattern was an isolated incident, the stream does not suffer much, as the multiplier if used only once. On the other hand, if there are repeated instances of a malicious pattern in the request stream, the multiplier is used repeatedly, leading to the stream being blocked much faster.

At this point, we make a distinction between two possible uses of the multiplier value. On one hand, the multiplier value can be maintained as a constant, which is used whenever a malicious pattern is detected. This scheme is called the Static Multiplier EDM. On the other hand, it is intuitive that the more a particular malicious pattern appears in a stream, the more possibility exists that it is malicious. Hence, the more prudent it is to block the stream faster. In order to do this, we maintain a list of observed malicious patterns and maintain a dynamic list of multiplier values. Each time a malicious pattern is observed, the multiplier value is decreased by a constant factor. Thus, the patterns which are observed more in request streams will have a lower multiplier value, which means that the streams which contain these patterns will be blocked much faster than others. This scheme is called the Dynamic Multiplier EDM, and can aid in detecting attacks much faster.

6.2. *Combining a PTA based ADM with EDM*

The automata based ADM works in two phases: learning and detection. In the learning phase, the ADM receives the system logs as input. The ADM computes the suspicion score values for all legitimate access logs, and fixes a lower bound for the value of suspicion scores. During the detection phase, the ADM assigns suspicion scores to all incoming connections, and if the suspicion score for any connection exceeds the identified threshold, it is blocked.

When the ADM detects any request stream to be malicious, it forwards the request stream to the EDM. The EDM extracts the repeating pattern of the stream as an attack signature and stores it. The EDM also continuously monitors the incoming request stream for the presence of any of the patterns stored in the malicious pattern dictionary. If the EDM detects the presence of a malicious pattern in the incoming request stream, it does not block the user immediately, as doing so would increase the false positive rate. Instead, it passes a multiplier (>1) which is used to reduce the observed sample entropy value for the request stream, so that it crosses the threshold faster. If the occurrence

Table 2
Descriptions of symbols used in Algorithm 2.

Symbol	Description
L	Legitimate Access Logs
A	Generated Attack Logs
S	State Set (Set of states within the application)
ExtractURL()	Function to extract URL from a given line of the access logs
MAX_ATTACK_LENGTH	Maximum number of attack requests generated in a session
CONDN	Condition used to select attack requests from the state set S

of the malicious pattern was an isolated incident, the stream does not suffer much, as the multiplier if used only once. On the other hand, if there are repeated instances of a malicious pattern in the request stream, the multiplier is used repeatedly, leading to the stream being blocked much faster.

An important point to note here is that the method of integrating the EDM for both the AL-DDoS detection mechanisms is virtually the same, which underlines the fact that the proposed EDM is firewall agnostic.

7. Results and discussion

A prototype of the proposed system was developed using the Go programming language. In order to test any AL-DDoS detection mechanism, access logs corresponding to normal, attack-free user traffic are required to perform the learning. Once the learning phase generates a model of legitimate user behaviour, testing is performed using access logs corresponding to AL-DDoS attacks. Since AL-DDoS attacks are highly specific to the web pages in a web application, it is mandatory that both the normal and attack traces correspond to the same web application. However, none of the existing datasets provide both legitimate and attack traces for the same web application. In order to test our system, we use open source datasets describing legitimate user access logs in the SDSC and CLARKNET websites. These traces are used to generate a model of legitimate user behaviour.

7.1. Dataset description

In order to test the efficiency of the system we have used two popular datasets which describe user access behaviour — SDSC-HTTP and CLARKNET-HTTP. SDSC-HTTP contains a day's worth of all HTTP requests to the SDSC WWW server located at the San Diego Supercomputer Center in San Diego, California, and contains 28,338 lines of access logs from 3555 clients. CLARKNET-HTTP is a much larger dataset, and contains two week's worth of all HTTP requests to the ClarkNet WWW server. It has 12,95,853 line of access logs from 77,431 clients and provides a suitable testing dataset. Each line in these access logs describes a request made by a client to the web application, and contain the same information — client address, date and time of access, request URL, status code and request size. Both of these datasets have been used extensively for evaluating AL-DDoS detection mechanisms [48,49].

7.2. Attack generation

A number of research works [50,51] have focused on generating AL-DDoS attack traces from legitimate traces in order to overcome this drawback. In this work, we generated our own attack traces using a similar approach, which is described in Algorithm 2. While generating attack vectors, we tried to include different varieties of attacks. A number of DDoS attacks target a single URL such as login pages. Such attacks are relatively easy to execute, but can be detected easily. The other strategy used by attackers is to use multiple URLs is a predefined sequence to evade detection. This attack requires knowledge about the structure of the web application, and hence requires more effort. However, the attacker is rewarded in the sense that such attacks are usually more difficult to detect. In addition to these two attack patterns, we introduce some more variations of attacks. Instead of treating all requests as the same, we made the following groupings:

Algorithm 2 Attack Log Generation
Input: Legitimate Access Logs, L
Output: Generated Attack Logs, A

```
$S \leftarrow \phi$
while True do
  Read next line of access log $L$ into $R$
  $curr\_state \leftarrow ExtractURL(R)$
  if $curr\_state \notin S$ then
    $S = S \cup \{curr\_state\}$
  end if
end while
$I = 1$
while $I < MAX\_ATTACK\_LENGTH$ do
  $urllist = SELECT\_RANDOM(S, I, CONDN)$
  $T = Current\_Time()$
  while True do
    $J \leftarrow 1$
    while $J < I$ do
      With probability p, generate log for request $R$ at time T
      With probability 1-p, generate log for request $R$ at time T+1
      $J \leftarrow J + 1$
    end while
    $I \leftarrow I + 1$
  end while
end while
```

- Base and Inline Requests: A web page is completely rendered only by the base HTML web page along with its constituent inline requests such as requests for image or CSS files. DDoS attacks can be carried out using either of the two types of requests, but attacks using base requests tend to be more effective [52,53].
- Request Workload: Every web request generates some amount of processing at the server side, or utilizes some server resources. Requests such as search queries require a larger amount of processing because they involve complicated joins and database searches. We call such queries high workload requests. High workload requests are used exclusively to launch asymmetric attacks.

Algorithm 2 describes the process of generating attack logs from legitimate access logs and Table 2 provides a description of the symbols used in Algorithm 2. The algorithm first constructs a state set S, which contains all the request URLs within the application. This is done by scanning through the legitimate access logs. Once the state set is constructed, the algorithm proceeds to select a subset of the states from the state set to launch an attack. The number of requests chosen from S will be less than the MAX_ATTACK_LENGTH. The states are chosen depending upon a specific condition, $CONDN$, which decides the type of attack generated. As an example, for generating asymmetric attacks, the condition would be that the workload of the states selected should be high. The algorithm generates attack logs corresponding to the selected states one after the other. The timestamps of the generated attack logs are the same as the previously generated attack log with a probability p. This generates logs which resemble real world DDoS attacks, with multiple requests being launched simultaneously without

Table 3
Types of attack generated.

Attack code	Attack type	Request type	Request workload	No. of URLs (Single/Multiple)
A1	Single URL Flood	Base	Any	Single
A2	Asymmetric Botnet	Base	High Workload	Multiple
A3	Single URL Asymmetric	Base	High Workload	Single
A4	Botnet Attack	Base	Any	Multiple
A5	Inline Flood	Inline	Low	Single
A6	Inline Bot	Inline	Low	Multiple
A7	Combined Bot Attack	Base + Inline	Any	Multiple

Table 4
Detection rate for application layer DDoS attacks using Sample Entropy.

Attack	SDSC DR (%)	CLARKNET DR (%)
A1	100	100
A2	95	95
A3	100	100
A4	100	100
A5	100	100
A6	100	100
A7	95	92

Table 5
Performance overview of attack detection module.

Parameter	SampEn based approach	PTA based approach
Detection Rate (DR)	99.7%	99.75%
False Positive Rate (FPR)	0.0025%	0.004%

Table 6
Attack-wise Performance of attack detection module.

Attack code	Detection rate using SampEn (%)		Detection rate using PTA (%)	
	SDSC	CLARKNET	SDSC	CLARKNET
A1	100	100	98	91
A2	95	95	100	100
A3	100	100	98	100
A4	100	100	100	100
A5	100	100	100	100
A6	100	100	100	100
A7	95	92	100	100

any time lag. Table 3 provides a summary of the different types of attacks generated.

7.3. Efficiency of detection

Each request stream is observed and the SampEn values are calculated. If the value of average SampEn falls below the threshold for any stream, that stream is labelled as an attack and blocked. It can be observed from Table 4 that SampEn performs extremely well in detecting application layer DDoS attacks, including asymmetric attacks. However, a detection mechanism using SampEn is unable to detect any attacks which use a random request generation mechanism for attack, because the stream will have a very high value of SampEn, comparable to legitimate request streams. The detection mechanism also has a very low false positive rate (FPR). The FPR in the case of SDSC and CLARKNET data were just 0.0005% and 0.0049% respectively.

Tables 5 and 6 compare the proposed SampEn based detection mechanism with the PTA based detection mechanism described by Praseed and Thilagam [17]. It can be seen that the proposed detection mechanism is able to detect AL-DDoS attacks with a high level of accuracy. In particular, the SampEn based detection mechanism displays a higher detection rate for some attack classes (A1 and A3). A model based detection mechanism under-performs in situations where some behavioural patterns were missed during the learning phase. In the PTA based detection mechanism, this corresponds to the learning phase not covering all possible combinations of states and transitions. This leads to the detection mechanism missing out some attack scenarios. A SampEn based detection mechanism, on the other hand, has a less stringent learning mechanism, and can detect attacks using states or transitions it has not previously seen. However, the less stringent detection mechanism also means that the SampEn based detection mechanism is unable to detect randomized AL-DDoS attacks. This is because random attack streams possess a high level of Sample Entropy, comparable to that of legitimate request streams. A model based detection mechanism, on the other hand, is able to detect randomized AL-DDoS attacks with considerable efficiency.

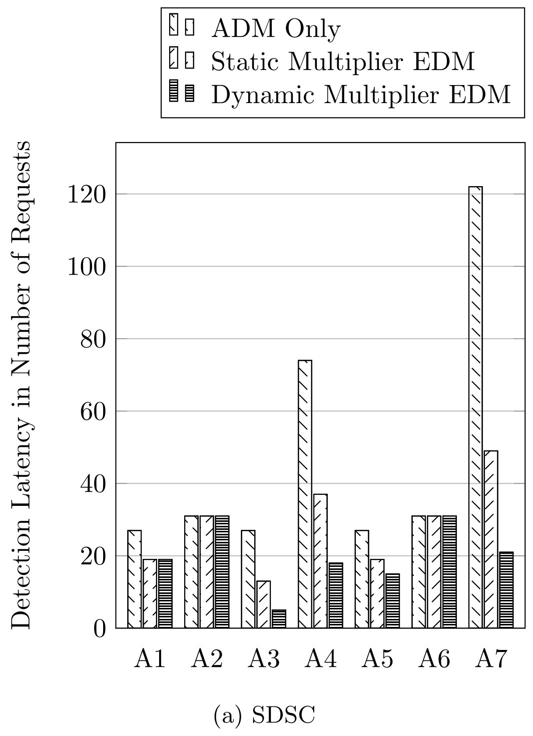
7.4. Impact of early detection

Figs. 5 and 6 shows that in all the cases, the latency of detection, measured in terms of the average decision length, decreases when EDM is used. This effect is more pronounced when a dynamic multiplier is used with the EDM. This clearly demonstrates that the use of a dynamic signature based detection mechanism in the form of the EDM speeds up the detection of application layer DDoS attacks. The use of an EDM cannot speed up detection for a randomized attack stream due to the absence of any repeating patterns.

Another important point related to the EDM is that it can be used with any anomaly based AL-DDoS detection mechanism with minimal modification. This further underlines the fact that the proposed EDM is firewall agnostic in nature.

8. Conclusion

Application layer DDoS attacks are one of the most prominent challenges in web security today. The low attack volume, coupled with a similarity to legitimate user traffic makes it immensely difficult to detect as well as block these attacks. However, these attacks do display a high degree of similarity both within a request stream and across attacking connections. This knowledge can be exploited to accelerate the detection of DDoS attacks using dynamic signature based approaches. In this work, we presented an Early DDoS Detection Module (EDM) which uses HTTP request patterns as signatures, which can work in combination with any existing anomaly based detection module (ADM). To demonstrate the efficiency of the EDM, we use an ADM which uses Sample Entropy of a request entropy for detecting application layer DDoS attacks. We showed that Sample Entropy proves to be a much more efficient indicator of an attack than Shannon's Entropy when URL sequences are considered. Using the Sample Entropy based ADM and an automata based ADM, we demonstrated that the use of the EDM considerably accelerates the process of detecting application layer DDoS attacks.



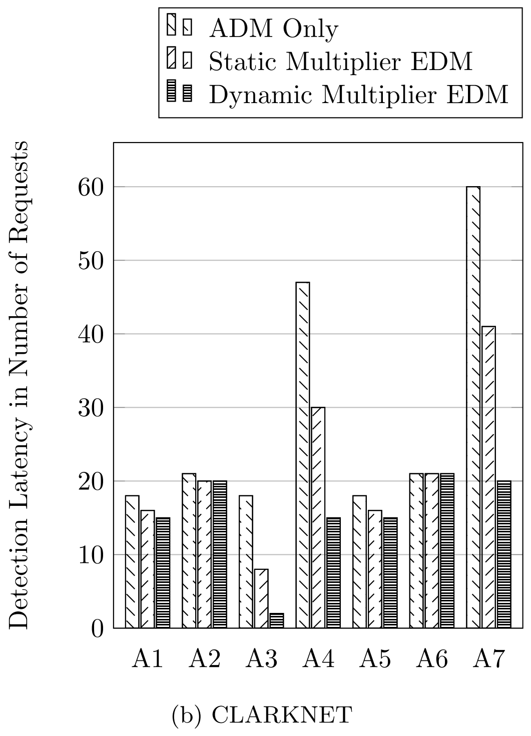
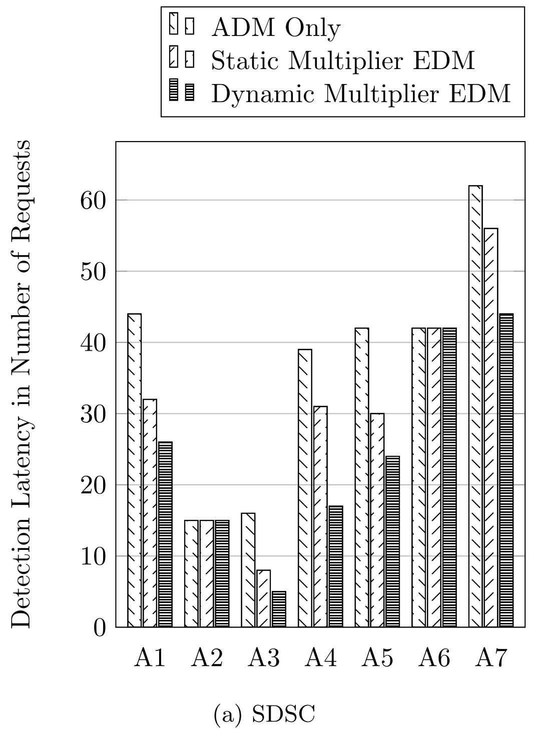


Fig. 5. Detection Latency for the SampEn based ADM.



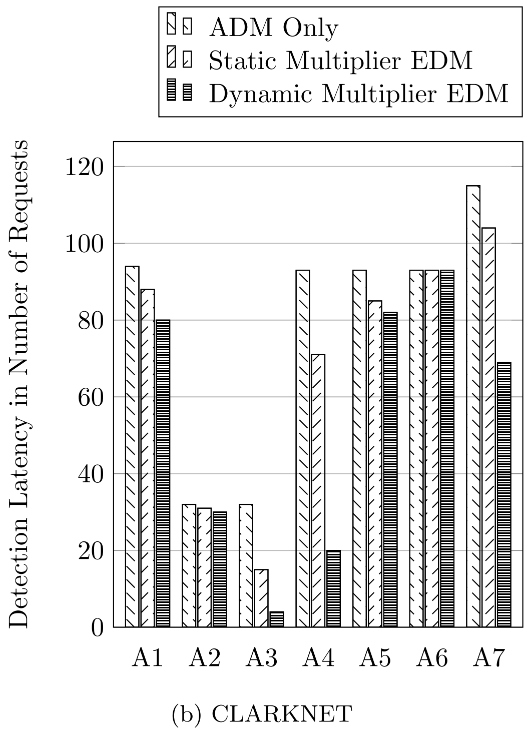


Fig. 6. Detection Latency for the Automata based ADM.

CRediT authorship contribution statement

Amit Praseed: Conceptualization, Methodology, Software, Writing – original draft. **P. Santhi Thilagam:** Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by the Ministry of Electronics and Information Technology (MeitY), Government of India and is part of the R&D project entitled "Development of Tool for Detection of Application Layer DDoS Attacks on Web Applications" during the period 2017–2020.

References

[1] Networks A. NETSCOUT Arbor's 13th annual worldwide infrastructure security report. 2019, URL https://pages.arbornetworks.com/rs/082-KNA-087/images/13th_Worldwide_Infrastructure_Security_Report.pdf.

[2] Antonakakis M, April T, Bailey M, Bernhard M, Bursztein E, Cochran J, Durumeric Z, Halderman JA, Invernizzi L, Kallitsis M, et al. Understanding the mirai botnet. In: USENIX Security Symposium; 2017, p. 1092–1110.

[3] Cochran J. The WireX Botnet: How industry collaboration disrupted a DDoS attack. 2017, URL https://blog.cloudflare.com/the-wirex-botnet/.

[4] Yu S, Zhou W, Jia W, Guo S, Xiang Y, Tang F. Discriminating DDoS attacks from flash crowds using flow correlation coefficient. IEEE Trans Parallel Distrib Syst 2011;23(6):1073–80.

[5] Labs D. Ddos ATTACKS against CAUCASIAN KNOT. 2019, URL https://equalit.ie/ddos-attacks-against-caucasian-knot/.

[6] Labs D. Ddos ATTACKS against vietnamese CIVIL SOCIETY. 2018, URL https://equalit.ie/ddos-attacks-vietnamese-civil-society/.

[7] Labs D. Botnet attack ANALYSIS OF DEFLECT PROTECTED website blacklivesmatter.com. 2016, URL https://equalit.ie/deflect-labs-report-3/.

[8] Singh K, Singh P, Kumar K. User behavior analytics-based classification of application layer HTTP-GET flood attacks. J Netw Comput Appl 2018;112:97–114.

[9] Katkar V, Bhirud S. Novel DoS/DDoS attack detection and signature generation. Int J Comput Appl 2012;47(10):18–24.

[10] Laskar S, Mishra D. Qualified vector match and merge algorithm (QVMMA) for ddos prevention and mitigation. Procedia Comput Sci 2016;79:41–52.

[11] Networks F. Detecting DoS attacks dynamically. 2019, URL https://techdocs.f5.com/kb/en-us/products/big-ip-afm/manuals/product/big-ip-system-dos-protection-and-protocol-firewall-implementations-14-0-0/07.html.

[12] Praseed A, Thilagam PS. Ddos attacks at the application layer: Challenges and research perspectives for safeguarding web applications. IEEE Commun Surv Tutor 2019;21(1):661–85. http://dx.doi.org/10.1109/COMST.2018.2870658.

[13] Bukac V, Matyas V. Analyzing traffic features of common standalone dos attack tools. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. Springer; 2015, p. 21–40.

[14] Nazario J. Blackenergy ddos bot analysis. Arbor Netw 2007.

[15] Liao Q, Li H, Kang S, Liu C. Application layer ddos attack detection using cluster with label based on sparse vector decomposition and rhythm matching. Secur Commun Netw 2015;8(17):3111–20.

[16] Welzel A, Rossow C, Bos H. On measuring the impact of ddos botnets. In: Proceedings of the Seventh European Workshop on System Security. ACM; 2014, p. 3.

[17] Praseed A, Thilagam PS. Modelling behavioural dynamics for asymmetric application layer DDoS detection. IEEE Trans Inf Forensics Secur 2020;16:617–26.

[18] Pincus SM. Approximate entropy as a measure of system complexity. Proc Natl Acad Sci 1991;88(6):2297–301.

[19] Resende PAA, Drummond AC. Http and contact-based features for botnet detection. Secur Priv 2018;1(5):e41.

[20] Tyagi R, Paul T, Manoj B, Thanudas B. A novel HTTP botnet traffic detection method. In: 2015 Annual IEEE India Conference (INDICON). IEEE; 2015, p. 1–6.

[21] Kwon J, Lee J, Lee H. Hidden bot detection by tracing non-human generated traffic at the zombie host. In: International Conference on Information Security Practice and Experience. Springer; 2011, p. 343–61.

[22] Behal S, Kumar K, Sachdeva M. D-FACE: An anomaly based distributed approach for early detection of ddos attacks and flash events. J Netw Comput Appl 2018;111:49–63.

[23] Xiang Y, Li K, Zhou W. Low-rate ddos attacks detection and traceback by using new information metrics. IEEE Trans Inf Forensics Secur 2011;6(2):426–37. http://dx.doi.org/10.1109/TIFS.2011.2107320.

[24] Kaushal K, Sahni V. Early detection of ddos attack in wsn. Int J Comput Appl 2016;134(13):0975–8887.

[25] Yu Chen, Kai Hwang. Collaborative change detection of ddos attacks on community and isp networks. In: International Symposium on Collaborative Technologies and Systems (CTS'06). 2006, p. 401–10. http://dx.doi.org/10.1109/CTS.2006.27.

[26] Yu S, Zhou W. Entropy-based collaborative detection of DDOS attacks on community networks. In: 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom). 2008, p. 566–71. http://dx.doi.org/10.1109/PERCOM.2008.12.

[27] Yu S, Zhou W, Doss R. Information theory based detection against network behavior mimicking ddos attacks. IEEE Commun Lett 2008;12(4):318–21. http://dx.doi.org/10.1109/LCOMM.2008.072049.

[28] Chen Y, Hwang K, Ku. W. Collaborative detection of ddos attacks over multiple network domains. IEEE Trans Parallel Distrib Syst 2007;18(12):1649–62. http://dx.doi.org/10.1109/TPDS.2007.1111.

[29] Dharmapurikar S, Lockwood J. Fast and scalable pattern matching for content filtering. In: 2005 Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE; 2005, p. 183–92.

[30] Huang K, Zhang D, Qin Z. Accelerating the bit-split string matching algorithm using bloom filters. Comput Commun 2010;33(15):1785–94.

[31] Aldwairi M, Al-Khamaiseh K. Exhaust: Optimizing wu-manber pattern matching for intrusion detection using bloom filters. In: 2015 2nd World Symposium on Web Applications and Networking (WSWAN). IEEE; 2015, p. 1–6.

[32] Choi B, Chae J, Jamshed M, Park K, Han D. {DFC}: Accelerating String Pattern Matching for Network Applications. In: 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16); 2016, p. 551–65.

[33] Jun J-H, Lee D, Ahn C-W, Kim S-H. DDoS attack detection using flow entropy and packet sampling on huge networks. Of: ICN; 2014, p. 185–90.

[34] Navaz A, Sangeetha V, Prabhadevi C. Entropy based anomaly detection system to prevent ddos attacks in cloud. 2013, arXiv preprint arXiv:1308.6745.

[35] Jie Zhang, Zheng Qin, Lu Ou, Pei Jiang, JianRong Liu, Liu AX. An advanced entropy-based DDOS detection scheme. In: 2010 International Conference on Information, Networking and Automation (ICINA). volume 2, 2010, p. V2–67–V2–71. http://dx.doi.org/10.1109/ICINA.2010.5636786.

[36] Kumar K, Joshi RC, Singh K. A distributed approach using entropy to detect ddos attacks in isp domain. In: 2007 International Conference on Signal Processing, Communications and Networking. 2007, p. 331–7. http://dx.doi.org/10.1109/ICSCN.2007.350758.

[37] Qin X, Xu T, Wang C. Ddos attack detection using flow entropy and clustering technique. In: 2015 11th International Conference on Computational Intelligence and Security (CIS). IEEE; 2015, p. 412–5.

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

[39] Zhao Y, Zhang W, Feng Y, Yu B. A classification detection algorithm based on joint entropy vector against application-layer DDoS attack. Secur Commun Netw 2018;2018.

[40] Johnson Singh K, Thongam K, De T. Entropy-based application layer ddos attack detection using artificial neural networks. Entropy 2016;18(10):350.

[41] Devi SR, Yogesh P. Detection of application layer ddos attacks using information theory based metrics. CS & IT-CSCP 2012;10:213–23.

[42] Jin Wang, Xiaolong Yang, Keping Long. A new relative entropy based app-DDoS detection method. In: The IEEE Symposium on Computers and Communications. 2010, p. 966–8. http://dx.doi.org/10.1109/ISCC.2010.5546587.

[43] Fan B, Andersen DG, Kaminsky M, Mitzenmacher MD. Cuckoo filter: Practically better than bloom. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. ACM; 2014, p. 75–88.

[44] Richman JS, Moorman JR. Physiological time-series analysis using approximate entropy and sample entropy. Am J Physiol-Heart Circ Physiol 2000;278(6):H2039–49.

[45] Rigoldi C, Cimolin V, Camerota F, Celletti C, Albertini G, Mainardi L, Galli M. Measuring regularity of human postural sway using approximate entropy and sample entropy in patients with ehlers–danlos syndrome hypermobility type. Res Dev Disabil 2013;34(2):840–6.

[46] Chen X, Solomon IC, Chon KH. Comparison of the use of approximate entropy and sample entropy: applications to neural respiratory signal. In: 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference. IEEE; 2006, p. 4212–5.

[47] Pan Y-H, Wang Y-H, Liang S-F, Lee K-T. Fast computation of sample entropy and approximate entropy in biomedicine. Comput Methods Programs Biomed 2011;104(3):382–96.

[48] Chwalinski P, Belavkin R, Cheng X. Detection of application layer ddos attacks with clustering and bayes factors. In: Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on. IEEE; 2013, p. 156–61.

[49] Chwalinski P, Belavkin R, Cheng X. Detection of application layer ddos attack with clustering and likelihood analysis. In: Globecom Workshops (GC Wkshps), 2013 IEEE. IEEE; 2013, p. 217–22.

[50] Lin C-H, Liu J-C, Chen C-R. Access log generator for analyzing malicious website browsing behaviors. In: 2009 Fifth International Conference on Information Assurance and Security. 2, IEEE; 2009, p. 126–9.

[51] Dhanapal A, Nithyanandam P. An effective mechanism to regenerate HTTP flooding ddos attack using real time data set. In: 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT). IEEE; 2017, p. 570–5.

[52] Ranjan S, Swaminathan R, Uysal M, Nucci A, Knightly E. DDoS-Shield: DDoS-resilient scheduling to counter application layer attacks. IEEE/ACM Netw 2009;17(1):26–39.

[53] Praseed A, Thilagam PS. Multiplexed asymmetric attacks: Next-generation DDoS on HTTP/2 servers. IEEE Trans Inf Forensics Secur 2019.

Amit Praseed received his B. Tech degree in Computer Science and Engineering in 2014 from College of Engineering Trivandrum, Kerala, India. He received his M. Tech degree in Computer Science and Engineering - Information Security in 2016 from National Institute of Technology Karnataka (NITK), Surathkal, India and received his Ph.D. degree in Computer Science from NITK in 2020. He is currently working as an Assistant Professor at Indian Institute of Information Technology (IIIT) Sri City. His research interests include Web Security and Information Security.

P. Santhi Thilagam received the B.E. degree in Computer Science and Engineering in 1991, and the M.E. degree in Computer Science and Engineering from College of Engg., Guindy, Anna University, Chennai, India, in 1999, and the Ph.D. degree in Information Technology from the National Institute of Technology Karnataka (NITK), Surathkal, India, in 2008. Since 1996, she has been with the Department of Computer Science and Engineering at NITK Surathkal where she is currently a professor. Her current research interests include database security, data management, data analysis, and distributed computing. Dr. Santhi is a member of several technical associations, scientific committees and editorial boards. She was the recipient of the best Ph.D. thesis award in Computer Science and Engineering Category of the Board of IT Education Standards in 2009, M.S Ramanujan Lecture presenter award of the Institution of Engineers India (IEI-India) in 2015.