# *ZeroWall*: Detecting Zero-Day Web Attacks through Encoder-Decoder Recurrent Neural Networks

Ruming Tang[†‡], Zheng Yang[♯‡], Zeyan Li[†‡], Weibin Meng[†‡], Haixin Wang[§],
Qi Li[♯‡], Yongqian Sun[¶], Dan Pei[*†‡], Tao Wei[‖], Yanfei Xu[£] and Yan Liu[£]
[†]*Dept. of Computer Science and Technology, Tsinghua University*
[‡]*Beijing National Research Center for Information Science and Technology (BNRist)*
[♯]*Institute for Network Sciences and Cyberspace, Tsinghua University*
[§]*University of Science and Technology Beijing*, [¶]*Nankai University*, [‖]*Baidu USA LLC*, [£]*Baidu, Inc*

***Abstract*—Zero-day Web attacks are arguably the most serious threats to Web security, but are very challenging to detect because they are not seen or known previously and thus cannot be detected by widely-deployed signature-based Web Application Firewalls (WAFs). This paper proposes *ZeroWall*, an unsupervised approach, which works with an existing WAF in pipeline, to effectively detecting zero-day Web attacks. Using historical Web requests allowed by an existing signature-based WAF, a vast majority of which are assumed to be benign, *ZeroWall* trains a self-translation machine using an encoder-decoder recurrent neural network to capture the syntax and semantic patterns of benign requests. In real-time detection, a zero-day attack request (which the WAF fails to detect), not understood well by self-translation machine, cannot be translated back to its original request by the machine, thus is declared as an attack. In our evaluation using 8 real-world traces of 1.4 billion Web requests, *ZeroWall* successfully detects real zero-day attacks missed by existing WAFs and achieves high F1-scores over 0.98, which significantly outperforms all baseline approaches.***

## I. Introduction

To defend against Web attacks, various defenses have been proposed [1]–[14] and adopted [15]–[17]. However, most of these approaches *cannot effectively detect zero-day Web attacks* [18], [19]. Zero-day attacks in general are hard to detect [20] and zero-day *Web* attacks are particularly challenging to detect for the following reasons.

First, since zero-day Web attacks have not been previously seen, most supervised approaches are inappropriate since these methods always require labeled data for training [9], regardless of signature-based Web Application Firewalls (WAFs) [15], [16] wildly deployed in industry [17] (*e.g.*, AWS WAF by Amazon [21], Yundun WAF by Alibaba [22] and Silverline by F5 Networks [23]) or machine-learning models [6], [7], [24], [25]. In other words, *unsupervised* approaches are more suitable for zero-day attack detection. Second, a Web attack can be carried out by a single malicious HTTP request. That is, those approaches [10], [11] utilizing contextual information between requests are not really helpful to detect such attacks. Thus we should focus on utilizing the syntax and semantics in individual requests for Web request detection. Third, zero-day Web attacks are very rare within a large number of Web requests [20]. Thus those unsupervised approaches based on collective and statistical information [1], [9], [13], [14] are not effective in detecting zero-day Web attacks.

This paper proposes *ZeroWall*, an unsupervised approach, which can work with an existing WAF in pipeline, to effectively detecting a zero-day Web attack hidden in an individual Web request. The key observation behind *ZeroWall* is that a benign Web request is a string following the HTTP protocol, *which can be treated as one sentence in the "HTTP request language"*, while a malicious Web request does not have consistent syntax and semantic patterns with the benign one. Our approach is inspired by the unsupervised self-translation machine [26] based on encoder-decoder recurrent neural networks: when trained with enough sentences in one language $A$, the neural network "understands" this language well enough such that it can translate an input sentence in $A$ into a latent representation which is in turn translated back into an output sentence in $A$. Intuitively, such a trained network can tell if a new sentence $s$ in unknown language belongs to language $A$ or not. Thus, if the translation quality is high, $s$ belongs to language $A$; otherwise, $s$ does not belong to language $A$.

*ZeroWall* can effectively detect zero-day Web attacks by mapping the zero-day Web attack detection problem to machine translation quality assessment problem. In *ZeroWall*, historical Web requests allowed by an existing WAF, a vast majority of which are benign, are used to train an encoder-decoder recurrent neural network that captures the syntax and semantics patterns of benign requests. The network "translates" a benign input Web request to latent representation (by the encoder) and then "translates" the representation back to an output Web request (by the decoder) which is close to the original request. In real-time detection, a zero-day attack request (which the WAF fails to detect), not understood well by the self-translation machine, cannot be translated back to its original request, thus is treated as an attack.

The contributions of this paper are the following.

- This paper advocates a general framework of augmenting existing signature-based WAFs with unsupervised machine learning based zero-day Web detection approach. This approach is immediately deployable and widely-applicable in real-world.
- To the best of our knowledge, this paper is the first to formulate the zero-day Web attack detection problem into

[*] Dan Pei is the corresponding author.

a neural machine translation quality assessment problem, a direction along which we believe more sophisticated solutions can be invented.
- We prototype *ZeroWall* by adopting the state-of-art neural machine translation algorithms, i.e., encoder-decoder recurrent neural networks. In our evaluation based on 8 real-world traces over 1.4 billion requests together, *ZeroWall* achieves high F1-scores (over 0.98) on all traces, significantly outperforming existing approaches.

The rest of this paper is organized as follows. §II introduces the related work. §III describes our core idea and system overview. §IV presents the implementation details of each components. §V evaluates *ZeroWall*. §VI discusses our design choices and limitations. §VII concludes our work.

## II. Related Work

Anomaly-based Web attack detection methods [1], [2] are becoming more and more popular recently, because in theory, they can detect both known or zero-day attacks. These approaches have the potential to detect zero-day Web attacks since malicious requests (typically assumed to be rare thus exhibit different patterns from benign ones') can appear "anomalous" during realtime detection. Based on the types of anomalies these methods focus on, they can be divided into three categories [9]: Single packet based *point anomaly detection* (*ZeroWall* belongs to this category) [9], [27], context based *contextual anomaly detection* [9]–[12] and statistic based *collective anomaly detection* [1], [9], [13], [14]. Normally, the later two approaches are unable to detect zero-day attacks because zero-day Web attack packets are not reflected in contexts and statistics. Thus, in this paper, we focus on point anomaly detection.

Point anomaly refers to a particular data instance different from normal ones [9]. That is, a single HTTP request in our case. One naive method is to split one HTTP request "sentence" into words. Intuitively, the collection of words used in malicious requests should be somehow different from that those word collections in benign requests. Such differences could be one indicator to detect the attacks [27]. However, more experienced attackers will use more covert ways to bypass such simple detection mechanism. For instance, the word collection of an attack request can be close to that of a benign request, if the attack request has a large amount of benign content in its request message.

Wang *et al.* presented PAYL using n-gram to inspect the simple histograms of payloads [3]. Ingham *et al.* presented a DFA-based approach to learn representations from requests [28]. Both approaches work well on simple requests, but cannot handle the various types of requests nowadays. Song *et al.* presented a hidden Markov model (HMM) called Spectrogram [4] to reconstruct content flows and extract features from packets. Ariu *et al.* presented Hmmpayl [5], an HMM-based IDS to detect attacks from payloads. However, HMM models work relatively worse when the length of the sequence is not suitable [5], which results in bad performance when processing complex requests. In contrast with above HMM-based approaches, our solution uses a neural network based method, which generally has better model capacity and better performance on complex requests.

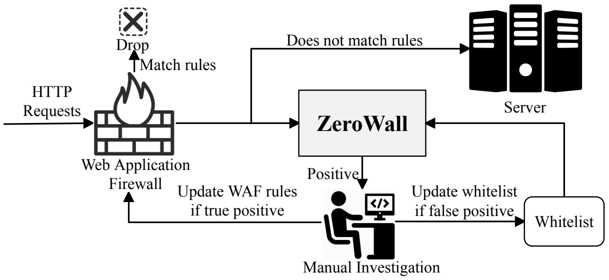

Fig. 1: The workflow of *ZeroWall*.

Deep learning-based methods were proposed to learn on complex HTTP request logs [6], [7], [24]. Naoum *et al.* presented an optimization to the back propagation in a neural network for IDS [24]. Srivastav *et al.* presented multiple Deep Learning-based methods for different types of attacks, respectively [6]. Zhang *et al.* proposed a convolutional neural network model to train on the words in the requests [7]. These works are all supervised, thus are not suitable for detecting zero-day attacks as mentioned in §I.

Vartouni *et al.* presented an unsupervised method using n-gram and stacked auto-encoder [29] to learn on HTTP requests [8]. However, this work directly uses the encoder output (dimension reduction) results for anomaly detection. Such outputs from anomalous samples may be indistinguishable from normal ones, which results in limited performance [30]. Our solution uses encoder-decoder network to reconstruct the sequences from the encoder output, *i.e.*, utilizing the decoder output, which performs better in practice (see §V).

## III. Core Idea and System Overview

In this section, we present the design goals, core idea and architecture of *ZeroWall*. Details will be presented later in §IV.

### A. Design Goals

As mentioned in §I, given the popular deployment of signature-based WAFs, it is better that the zero-day Web attack detection mechanism can work together with the existing WAFs, rather than entirely replacing WAFs. This is our first design goal. As shown in Fig. 1, *ZeroWall* is used as a bypass system behind WAF, which does not cause additional overhead to the Web services. Arriving requests matching some WAF rules will be dropped by the WAF, effectively detecting those known attacks; those unmatched rules are then passed along to servers, and the traffic is mirrored and passed to *ZeroWall*, which focuses on detecting zero-day attacks. Since commonly seen malicious samples are not friendly to unsupervised algorithms, it is another benefit from working together with WAFs. The existing WAFs will filter out known attacks, which will improve the performance of the unsupervised algorithm. This is also a good solution to defend evasions like poisoning attacks, which will inject large numbers of attacks in the traffic.

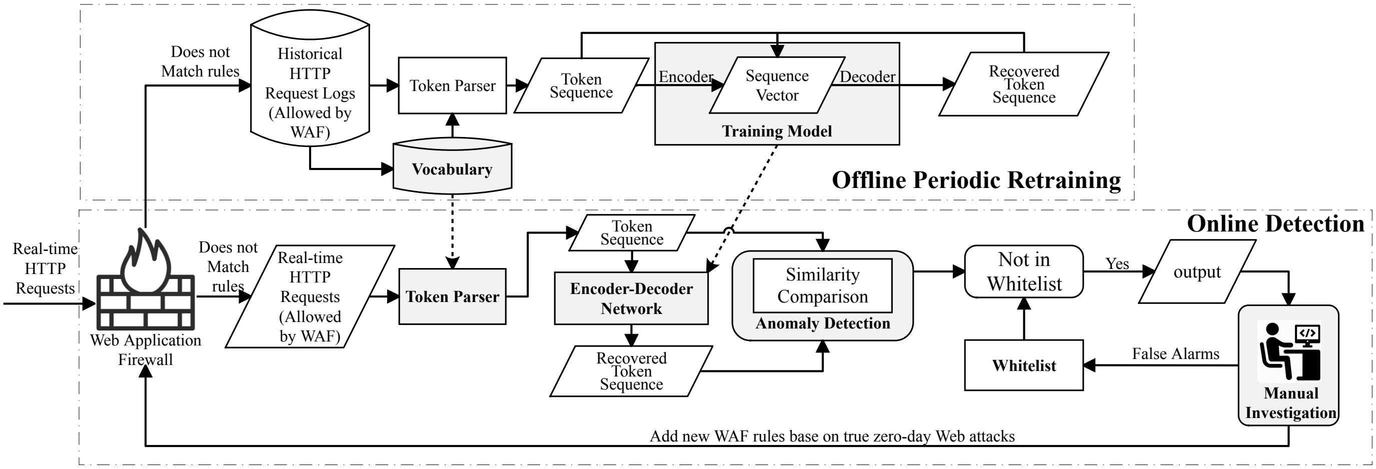

Fig. 2: The overview of components in *ZeroWall*.

Our second design goal is to detect zero-day Web attacks in HTTP requests through analyzing each single request.We chose to use an unsupervised method, for two reasons. First, an unsupervised approach does not have to know beforehand the exact patterns of attacks, thus it has much more potential to detect zero-day attacks than supervised ones. Second, in the HTTP requests arriving at the WAF, the benign to malicious ratio is typically very high, and this ratio gets even much higher (2811:1∼88863:1 in our traces) after the WAF drops those known attacks based on rules (See Fig. 1). Such an extreme class imbalance is very challenging to supervised approaches, but is naturally beneficial to unsupervised approaches.

We advocate the above general framework of augmenting existing signature-based WAFs with unsupervised machine learning based zero-day Web attack detection approach. This detection approach is immediately deployable and widely-applicable in real-world.

TABLE I: One Example of Benign Request.

| | | | |
|---|---|---|---|
| **Original Request** | POST http://A.Example.COM/URL_1/URL_2/JSP_NAME_A.jsp modo=entrar&login=HASHED_NAME&pwd=HASHED_PWD&remember=off&B1=Entrar | | |
| **Token Sequence** | _url_1_ _url_2_ JSP_NAME_A jsp modo entrar login _OTHER_ pwd _OTHER_ remember off b1 entrar | | |
| **Recovered Token Sequence** | _url_1_ _url_2_ JSP_NAME_A jsp modo entrar login _OTHER_ pwd _OTHER_ remember on b1 entrar | | |
| **BLEU** | 0.8091 | **Malicious Score** | 0.1909 |

TABLE II: One Example of Malicious Request.

| | | | |
|---|---|---|---|
| **Original Request** | POST http://B.Example.COM/URL_3 /images/image/ASP_NAME_B.asp/FILENAME_B.jpg 0=M&z0=GB2312&z1=/ccmd&z2=echo 'phpinfo' | | |
| **Token Sequence** | _other_ images image ASP_NAME_B asp _other_ jpg _pnum_ _onechr_ z_pnum gb2312 z_pnum _other_ z_pnum echo phpinfo | | |
| **Recovered Token Sequence** | _other_ images images images images images _other_ images _hostname_ com callback pfk cqdbyy pfk com jpg | | |
| **BLEU** | 0.147342 | **Malicious Score** | 0.852658 |

### *B. Core Idea of Unsupervised Detection*

Our unsupervised zero-day Web attack detection approach is based on anomaly detection (as mentioned in §II). The training data of *ZeroWall* are those historical Web requests that do not match any WAF signature rules (see the top left of Fig. 2). We use encoder-decoder recurrent neural networks as our base training algorithm, which are widely used in modeling sequential data [31], including machine translation, speech recognition, questions answering, *etc.* The key observation behind *ZeroWall* is that an HTTP request is a string following the HTTP protocol, and *we can consider an HTTP request as one sentence in the "HTTP request language"*. Our actual solution is inspired by the unsupervised self-translation machine [26] based on encoder-decoder recurrent neural networks: when trained with enough sentences in one language $A$, the neural network "understands" this language well enough such that it can translate an input sentence in $A$ into a latent representation, which is in turn translated back into an output sentence in $A$. The goal of the neural network training is to make the output sentence close to the input sentence enough. Such a trained network can be used to tell whether a new sentence $s$ belongs to language $A$ or not. If the translation quality of $s$ is high, $s$ belongs to language $A$; otherwise, $s$ does not belong to language $A$. In other words, the translation quality of zero-day Web attacks (which are very rare) should be much worse than that of benign requests, thus we can use a translation quality threshold to declare the attacks.

To the best of our knowledge, we are the first to formulate the zero-day Web attack detection problem into a neural machine translation quality assessment problem, a direction along which we believe more solutions can be invented.

### *C. System Overview*

As shown in Fig. 2. *ZeroWall* can be divided into two phases: *offline periodic retraining* and *online detection.*

*1) Offline Periodic Retraining:* The offline training is conducted periodically (*e.g.*, daily) or triggered manually to utilize the latest request data filtered by the latest WAF rules (*e.g.*, new deployed APIs cause unseen requests). Each round of offline retraining phase utilizes the historical dataset of HTTP request logs (allowed by the latest WAF rules) collected so far by the time of this round of retraining (called the retraining dataset hereinafter in the paper), and has three steps:

1) **Building Vocabulary**: *ZeroWall* uses tokenization techniques to extract words from the retraining dataset, filters unnecessary words (*e.g.*, stop words), and then applies *word embedding* technique to represent words.
2) Then the **Token Parser** component of *ZeroWall* uses the vocabulary to convert each HTTP request in the retraining dataset into a token sequence (Table I and II show two examples, see the second rows).
3) **Training Model**: *ZeroWall* trains the Encoder-Decoder neural network using previously generated sequences.

When the training is complete, the vocabulary and model will be used as two components in online detection phase.

*2) Online Detection:* During **online detection**, *ZeroWall* will detect in real-time whether an HTTP request is benign or malicious (*a.k.a*, zero-day attacks in our dataset behind the WAF). In this phase, *ZeroWall* has four major components: Token Parser, Encoder-Decoder Network, Anomaly Detection, and Manual Investigation.

The first two components are the results of offline training phase. Given one HTTP request as input, *ZeroWall* will first tokenize the request into one *token sequence* (see the second rows of Tables I and II for examples) using the vocabulary built offline. One recurrent neural network (encoder) is used to read the token sequence and build the corresponding sequence vector, a sequence of numbers that represent the token sequence meaning. Then we use another recurrent neural network (decoder) in order to reconstruct the *recovered token sequence* (see the third rows of Tables I and II for examples) according to the sequence vector. This is often referred as the encoder-decoder architecture [32]–[34].

The third component is the **Anomaly Detection**. Our idea for anomaly detection is inspired by the quality assessment method in machine translation. That is, if the recovered token sequence is similar to origin token sequence, then the request is considered benign. Otherwise, it is malicious and *ZeroWall* will mark it and report to security engineers. In this paper, we choose BLEU metric [35] (out of several alternatives, as will be shown in Fig. 6) as the indicator of the similarity between original and recovered token sequences (see the last rows of Tables I and II for examples). BLEU is a metric used to determine the similarity between sentences, and is widely used in machine-translation problems [36], [37] for comparing the machine translation results to manually translated sentences.

The fourth component is the **Manual Investigation**. Unsupervised anomaly detection cannot entirely avoid false positives. For example, some new requests with some patterns very different from all previously seen patterns might be perfectly benign. Thus, the requests declared as "malicious" will be investigated manually by security engineers (see the bottom of Fig. 1 and the top right of Fig.2). False alarms confirmed by security engineers are incorporated into a whitelist, which is used to avoid future false positives caused by the requests with same patterns. Such whitelist can improve the performance of *ZeroWall*, which will be discussed in §V-G. True zero-day attacks confirmed by the security engineers are used to compose new signature-based rules which are incorporated into the WAF (see the left of Fig. 1 and Fig. 2). The overhead of manual investigation is acceptable given the small number of true zero-day attacks (because they are rare in practice) and false positives (see §V).

In summary, the online detection phase has the following steps to process one HTTP request:

1) **Token Parser**: Convert one original HTTP request into a token sequence using the vocabulary built offline.
2) **Encoder-Decoder Network**: Reconstruct the recovered token sequence by the neural network trained offline.
3) **Anomaly Detection**: Calculate the BLEU score of two sequences and compare it to a threshold to decide whether the request is benign or malicious, and then report the output to security engineers.
4) **Manual Investigation:** Incorporate manual confirmed false alarms and true zero-day attacks into whitelist and WAF, respectively.

Note that, since the volume of incoming traffic is extremely large, machine learning based Web attack detection approaches may not be able to process all the traffic. In order to address this issue, we leverage hash tables to speed up the processing of pre-parsing. For each incoming token sequence, we calculate its hash value to efficiently verify if it is already been processed previously. Also, we build a table to store all processed information to avoid repeated calculation. This method significantly reduces the process overhead so that it can process all traffic. Actually, we can use multiple front-end servers to do the pre-parsing tasks concurrently while collecting the data, which will be our future work.

## IV. Design Details

### A. Token Parser

As mentioned before, the very first step of *ZeroWall* is to convert these strings into token sequences. In this paper, we consider one HTTP request as a string consisting of words. We first need to extract all the useful words (*a.k.a*, tokens) out of each given request. Then, we use the entire set of tokens (*a.k.a*, *vocabulary*) to parse each string into a *token sequence*.

*1) Vocabulary: ZeroWall* first splits the entire set of request strings with punctuation and space. After that, we will get multiple "words" from the strings. However, not all of words will be included in the vocabulary, and we need to filter useless and meaningless ones. One kind of these useless words are variables. Their values do not count much in the training and should be replaced by placeholders (*e.g.*, *_uid_*). The other kind of these useless candidates are stop words (*e.g.*, *the, and*). In spite of their high frequencies, these candidates provide little information for the requests. As a result, these words are also replaced by placeholders when building the vocabulary. Formally, we filter the words according to their frequencies. Words with too low or too high frequencies will be ignored, and the set of the remaining words is called *vocabulary*. In the vocabulary, each word is associated with a token-ID.

*2) Token Sequence:* When vocabulary building is done, we can convert a request into a token sequence. This step is done by *Token Parser*. Simply, it only keeps the words which are in the vocabulary and filters other words. The examples of token sequence are shown in the second row in Table I and II. We can see some variables are replaced by placeholders.

After the steps above, we are able to convert an original HTTP request string into a sequence of tokens, called *Token Sequence* or *Original Token Sequence* in this paper.

*3) Token Embedding:* Note that each token is represented by its ID in the vocabulary, and the token-ID neglects the meanings of the token. Therefore, we add an embedding layer to map these token-IDs into latent vectors, using vector distances between tokens to represent their logic relationships. This is also well known as *word embedding* or *word2vec* [38].

### *B. Encoder-Decoder Network*

The core function of encoder-decoder network is to map one sequence to another sequence. In this paper, we use token sequences obtained from above steps (called *original token sequence*) as inputs. The outputs (called *recovery token sequence*) are the sequences reconstructed by the encoder-decoder network after learning on data.

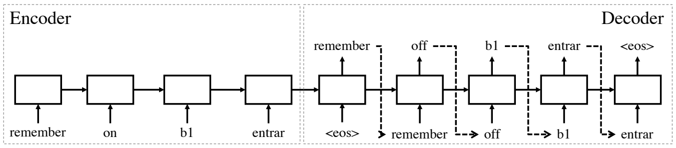

Fig. 3: Encoder-decoder network. The example sequences used here are parts of the sequences shown in Table I.

Fig. 3 shows the typical structure of encoder-decoder network described in [33] and one example. Given one input sequence “remember on b1 entrar *EOF*”, the network will output another sequence “remember off b1 entrar *EOF*”. Please note that these sequences here are just examples and the results may be different in the wild, and the lengths of the two sequences may not be the same.

In *ZeroWall*, the **encoder** will read the token sequences as inputs. Fig. 4 shows how the encoder-decoder works. The token sequence (the first column in Fig. 4, also represented as $[33, 0, 79, 0, 4, 66, 52, 33]$, these numbers are token-IDs in the vocabulary) is sent to an embedding layer, which further converts the tokens into latent vectors, moving them to the encoder network. The encoder network is one LSTM network [39], it computes the representation (a fixed-dimensional vector) of the original sequence.

The **decoder** is another LSTM network which uses the representation to compute the probability of output sequences. The output of the decoder is several $1 \times N$ vectors (where $N$ is the number of tokens in vocabulary) (see the third last column in Fig. 4), each element in a vector represents the probability of a token. For each output vector, the decoder will choose the token with the highest probability. After these we can get the recovered token sequences using the same vocabulary (see the last column in Fig. 4). The overall processing can be described as reconstruction of the original token sequence based on the learning of the network. That is, the neural network is trying to reconstruct the token sequence with tokens in the same vocabulary which represents the same information.

In the example presented in Fig. 4, we could get the recovered sequence *“onechr _OTHER_ do _OTHER_ userid _pnum_0_ _pbas_0_ _pnum_1_”*. We can see clearly that the original token sequence and the recovered token sequence are different (especially the last halves). This example is one malicious request from our dataset. This is because the encoder-decoder network has not seen this request preciously, and thus it cannot reconstruct it well. Meanwhile, in the benign request example in Table I, the recovered token sequence (third row) is obviously similar to the original one (second row).

Here we reiterate our observation: in practice, proportions of benign and malicious requests are extremely imbalanced in the HTTP logs allowed by the WAF (our training set), *i.e.*, a vast majority of Web logs are benign requests. As a result, malicious requests should have little impact on the learning of encoder-decoder network in our system, as confirmed later in our evaluation in §V.

### *C. Anomaly Detection*

As mentioned before, the core idea of encoder-decoder network is mapping one sequence to another sequence based on the “understanding” of these sequences. The encoder-decoder network is able to reconstruct the benign token sequences with higher accuracy. On the other hand, when the network takes a zero-day Web attack request as input (which is rare in the dataset), the output will be much more different from the input.

To do the anomaly detection, if the input is commonly seen in the learning data, the result will be good; if the input is an outlier, the result is probably bad. To judge the result of this mapping, *i.e.*, the learning of HTTP requests, we can compare the similarity between the token sequence and the recovered token sequence. Then we use it as an anomaly indicator in the detection. The workflow of these steps are shown in Fig. 5. We use BLEU [35] metric (we justify the selection of the BLEU metric later in §VI) to calculate the similarity in this paper.

The examples given above in Table I and II show different outputs of the encoder-decoder network. Given some inputs, the recovery token sequences are similar to the input ones, while sometimes the recovered sequences are much more different from the original ones. To access the quality of the sequence to sequence mapping, we use BLEU metric here. Simply, BLEU metric ranges from 0 to 1, a higher value indicates more similarity between source sequence and target sequence [35]. For our purpose, we define *malicious score*, $1 - BLEU$, as our estimator. A large malicious score means the encoder-decoder network is unable to reconstruct the given sequence, indicating that the input sequence may be malicious. As shown in Fig. 5, the malicious scores will be used to compare with a certain threshold, to tell whether the requests are malicious or benign for further operation.

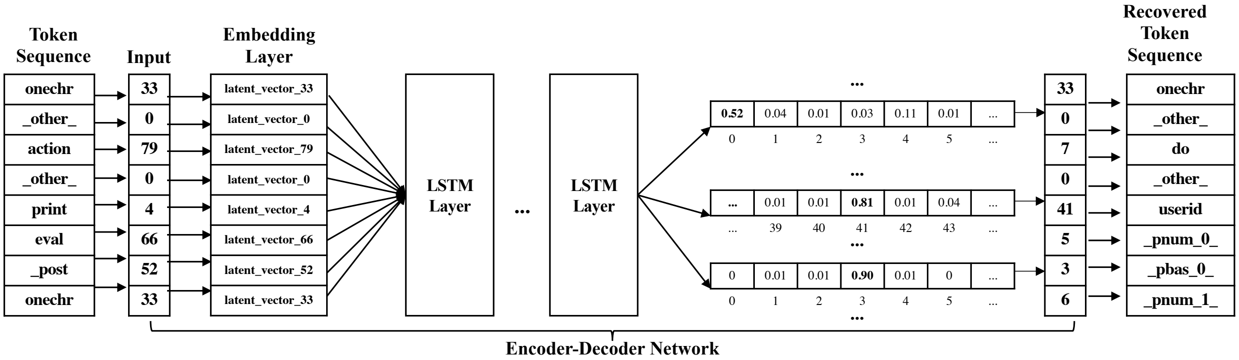

Fig. 4: An example of the encoder-decoder network. The input and output sequences are those shown in Table III, respectively.

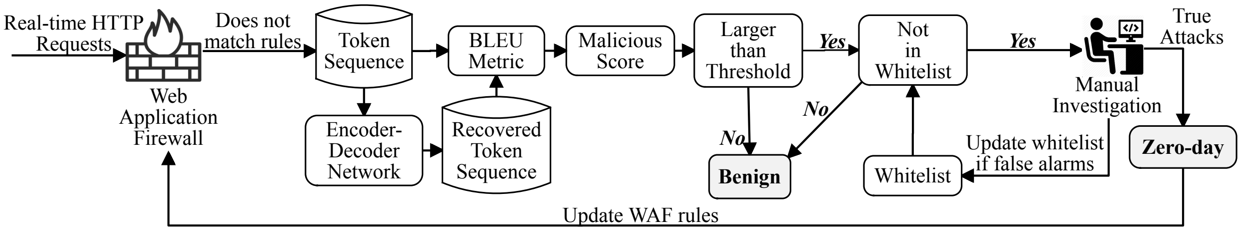

Fig. 5: The workflow of anomaly detection and manual investigation in *ZeroWall*.

TABLE III: An example of malicious requests through the encoder-decoder network and detection step. (TS: token sequence; RTS: recovered token sequence)

| | | | |
|---|---|---|---|
| **TS** | onechr _OTHER_ action _OTHER_ **print eval _post onechr** | | |
| **RTS** | onechr _OTHER_ do _OTHER_ **userid _pnum_0_ _pbas_0_ _pnum_1_** | | |
| **BLEU** | 0.4258 | **Malicious Score** | 0.5741 |

The examples we used in Table I and II also present the BLEU values and malicious scores of the two requests. We can see in Table I that the recovered token sequence is almost the same as original token sequence. This leads to a high value of BLEU metric (0.81) and small malicious score (0.19). In Table II, a malicious request ends up with more differences in the recovered token sequences, leading to a BLEU value of 0.15 and a malicious score of 0.85.

Table III represents another example. The first half of original token sequence is normal and harmless, while the rest is clearly probing attempt. Comparing the original and recovered token sequences, we could find that first halves of the two sequences still share some similarity. However, the rest parts are obviously different. Since the trained model is unable to rebuild the "original" token based on its knowledge of benign requests, this result in relative smaller BLEU value, *i.e.*, larger malicious score. The result will be fed into the component of Manual Investigation for verification and used to update the **whitelist** and WAF, respectively (see §III-C).

## V. Evaluation Using Real-World Traces

In this section, we evaluate *ZeroWall*'s zero-day Web attack detection performance using 8 real-world traces consisting of large scale of HTTP requests, and compare its performance with those of representative baseline approaches from §II.

### A. Baseline Approaches

As mentioned in §II, the approaches focusing on contextual and collective anomaly detection are inappropriate in our scenario, thus are not compared. We picked the following representative approaches from point attack detection category (which *ZeroWall* falls into). These approaches can be further classified into two types. Stacked auto-encoder approach [8] (named as *SAE*), HMM-based approach [5] (named as *Hmm-payl*) and DFA-based approach [28] (named as *DFA*) are **unsupervised** approaches. CNN-based *CNN-Token* approach [7], and RNN-based *RNN-Token* approach [25] are **supervised** approaches. We also implemented another supervised approach, *DT-Token*, based on a simple but popular supervised classifier decision tree. *DT-Token* takes all tokens in a Web request as features and outputs the prediction results.

### B. Real-World Traces

We obtained real-world traces from a WAF $W$ operated by a top global Internet company $I$ which offers signature-based WAF service to hundreds of its enterprise customers. We collected 8 seven-day traces during the first week in June, 2019. Each trace (denoted by $D$-$n$) collected the full set of requests of a specific enterprise customer, whose diverse statistics are shown in the first column in Table IV. Furthermore, the types of requests (APIs of different services) in the traces are over $8,000$, and the parameters vary in numbers and types. Therefore, overall these traces offer a good diversity for evaluating ZeroWall' s robustness across different customers.

### C. Labels, Training, and Testing

*1) Obtaining Labels:* The traces we used are collected behind the WAF. The requests matching WAF rules are dropped by WAF (called *WAF-malicious requests* in this section), others are allowed by WAF. Note that a WAF with imperfect rules in practice cannot guarantee to drop *all* known Web attacks. For a request allowed the WAF, if it is detected as a zero-day attack by at least one of the *ZeroWall* or baseline approaches *and* confirmed by the security engineers, it is called a "zero-day Web attack"; otherwise a "benign request". The first column of Table IV shows the counts of these requests, while B2M and B2Z mean the *benign* to *WAF-malicious* ratio and the *benign* to *zero-day* ratio, respectively.

*2) Training and Testing:* We split each 7-day trace into two parts according to time: the first day is used for training (training set), and rest is used as real-time traffic in online detection (testing set). These datasets vary in volumes and ratios. Based on these, we can determine the performance of *ZeroWall* in different real-world situations.

During training, unsupervised approaches (including *ZeroWall*) use all requests allowed by the WAF, without using any labels; while supervised approaches use all requests (allowed and dropped) and use the WAF's detection results (allowed/dropped) as labels. The inputs to all approaches are the token sequences generated by the Token Parser component of *ZeroWall*, instead of the original HTTP requests.

During testing, we focus on only zero-day Web attacks, and do not consider known attacks (which are already dropped by WAF). Thus in testing, the *positive samples refer to the zero-day attack requests in the traces*, defined in §V-C1, while the negative samples are benign requests.

### D. Ground Truth and Evaluation Metrics

We evaluate different approaches' performance using three metrics: $precision$, $recall$, and $F1\text{-}score$. For each approach, we first calculate true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). True positives are those zero-day attack requests in the dataset that are correctly detected as zero-day attacks, while true negatives are those benign requests that are correctly detected as benign. False positives are those benign requests that are detected as zero-day attacks, while false negatives are those zero-day attack requests that are detected as benign. We then calculate metrics as follows: $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, and $F1\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall}$.

TABLE IV: Statistics and evaluation results on 8 traces.

| Trace | Approach | Precision | Recall | F1-Score |
|---|---|---|---|---|
| **D-1** | ***ZeroWall*** | **0.9855** | **1.0000** | **0.9889** |
| #WAF-Malicious: 51,839 | *DT-Token* | 0.0010 | 1.0000 | 0.0019 |
| #Zero-Day Attacks: 25 | *CNN-Token* | 0.0010 | 1.0000 | 0.0019 |
| #Benign: 1,576,235 | *RNN-Token* | 0.0000 | 1.0000 | 0.0000 |
| #Total: 1,628,099 | *SAE* | 0.0001 | 1.0000 | 0.0002 |
| B2M: 30.4 | *Hmmpayl* | 0.0000 | 0.0000 | 0.0000 |
| B2Z: 63049.4 | *DFA* | 0.0000 | 1.0000 | 0.0000 |
| **D-2** | ***ZeroWall*** | **1.0000** | **1.0000** | **1.0000** |
| #WAF-Malicious:186,066 | *DT-Token* | 0.0547 | 0.3712 | 0.0931 |
| #Zero-Day: 1,118 | *CNN-Token* | 0.3300 | 0.7784 | 0.4593 |
| #Benign: 3,142,793 | *RNN-Token* | 0.0005 | 0.9760 | 0.0010 |
| #Total: 3,329,977 | *SAE* | 0.0005 | 0.9820 | 0.0010 |
| B2M: 16.9 | *Hmmpayl* | 0.0000 | 0.0000 | 0.0000 |
| B2Z: 2811.1 | *DFA* | 0.0004 | 1.0000 | 0.0008 |
| **D-3** | ***ZeroWall*** | **0.9925** | **0.9687** | **0.9805** |
| #WAF-Malicious: 19,515 | *DT-Token* | 0.7388 | 0.2463 | 0.3658 |
| #Zero-Day: 283 | *CNN-Token* | 0.4230 | 0.6376 | 0.5039 |
| #Benign: 13,572,827 | *RNN-Token* | 0.0000 | 0.9999 | 0.0001 |
| #Total: 13,592,625 | *SAE* | 0.0015 | 0.9130 | 0.0030 |
| B2M: 695.5 | *Hmmpayl* | 0.0000 | 0.0000 | 0.0000 |
| B2Z: 47960.5 | *DFA* | 0.0000 | 1.0000 | 0.0001 |
| **D-4** | ***ZeroWall*** | **0.9879** | **1.0000** | **0.9939** |
| #WAF-Malicious: 53,394 | *DT-Token* | 0.0001 | 1.0000 | 0.0002 |
| #Zero-Day: 4,209 | *CNN-Token* | 0.0001 | 1.0000 | 0.0002 |
| #Benign: 15,618,518 | *RNN-Token* | 0.0008 | 1.0000 | 0.0016 |
| #Total: 15,676,121 | *SAE* | 1.0000 | 0.0000 | 0.0000 |
| B2M: 292.5 | *Hmmpayl* | 0.0000 | 0.0000 | 0.0000 |
| B2Z: 3710.7 | *DFA* | 0.0001 | 1.0000 | 0.0002 |
| **D-5** | ***ZeroWall*** | **0.9928** | **1.0000** | **0.9964** |
| #WAF-Malicious: 33,724 | *DT-Token* | 0.2497 | 0.0082 | 0.0153 |
| #Zero-Day: 1,188 | *CNN-Token* | 0.6567 | 0.5410 | 0.5883 |
| #Benign: 31,718,124 | *RNN-Token* | 0.9988 | 0.0328 | 0.0629 |
| #Total: 31,753,036 | *SAE* | 0.0000 | 0.0492 | 0.0000 |
| B2M: 940.5 | *Hmmpayl* | – | – | – |
| B2Z: 26698.8 | *DFA* | 0.0001 | 1.0000 | 0.0001 |
| **D-6** | ***ZeroWall*** | **1.0000** | **0.9897** | **0.9948** |
| #WAF-Malicious:2,136K | *DT-Token* | 0.1733 | 0.0365 | 0.0576 |
| #Zero-Day: 2,003 | *CNN-Token* | 0.0204 | 0.0590 | 0.0269 |
| #Benign: 177,993,528 | *RNN-Token* | 0.0000 | 1.0000 | 0.0000 |
| #Total: 180,132,342 | *SAE* | 0.0001 | 0.1461 | 0.0001 |
| B2M: 83.3 | *Hmmpayl* | - | - | - |
| B2Z: 88863.5 | *DFA* | 0.0000 | 1.0000 | 0.0000 |
| **D-7** | ***ZeroWall*** | **0.9943** | **1.0000** | **0.9971** |
| #WAF-Malicious:42,088K | *DT-Token* | 0.0874 | 0.0267 | 0.0377 |
| #Zero-Day: 49,011 | *CNN-Token* | 0.8094 | 0.3027 | 0.4366 |
| #Benign: 528,158,912 | *RNN-Token* | 0.6857 | 0.5608 | 0.6120 |
| #Total: 570,296,546 | *SAE* | 0.0001 | 0.5691 | 0.0002 |
| B2M: 12.5 | *Hmmpayl* | - | - | - |
| B2Z: 10776.3 | *DFA* | 0.0001 | 1.0000 | 0.0002 |
| **D-8** | ***ZeroWall*** | **0.9966** | **0.9983** | **0.9974** |
| #WAF-Malicious:90,982K | *DT-Token* | 0.2036 | 0.3054 | 0.2396 |
| #Zero-Day: 83,746 | *CNN-Token* | 0.2525 | 0.0275 | 0.0479 |
| #Benign: 534,048,878 | *RNN-Token* | 0.5237 | 0.0718 | 0.1242 |
| #Total: 625,115,143 | *SAE* | 0.0008 | 0.3476 | 0.0017 |
| B2M: 5.9 | *Hmmpayl* | - | - | - |
| B2Z: 6377.0 | *DFA* | 0.0000 | 1.0000 | 0.0005 |

The primary challenge of the evaluation using real-world traces is the vast number of requests that need to be labeled (at least partially) manually by security engineers to obtain the ground truth. Thus, the ground truth can be only *approximately* obtained: the requests detected as zero-day attacks by *ZeroWall* and all baseline approaches are checked and rectified (if necessary) manually by the security engineers. Note that it is possible that some zero-day attacks are still missed by all above approaches, but it is infeasible for the engineers to manually check all requests. Therefore we settled with the above approximate ground truth in our evaluation, with the understanding that the resulting recall might be artificially higher than the truth (should there be any zero-day attacks missed by all approaches, which are then FNs). But the resulting precision will not change, because all TPs and FPs are manually checked. Note that these zero-day attack ground truth labels are for *evaluation* purpose only. Unsupervised approaches do not use labels for training, and supervised approaches directly use the WAF output as labels (as opposed to using ground truth labels) for training.

Hyper-parameters (such as the anomaly detection threshold in *ZeroWall*) affect system's performance, we thus obtain and present the best F1-scores (across the hyper-parameter space) and its corresponding precisions and recalls.

### *E. Experiments Results*

The best F1-scores with its corresponding precisions and recalls are shown in Table IV. All these results are for zero-day attacks only. We observe that *ZeroWall* significantly outperforms all existing approaches, achieving the best F1-scores over 0.98 on all 8 traces. The precision and recall values are also very high (almost 0.99 and 1).

These 8 traces are different in volumes, numbers of types of requests and data distributions. Among all approaches, *ZeroWall*'s variances across different traces are most stable (*i.e.*, has the smallest gaps between maximum and minimum).

Supervised approaches *DT-Token*, *CNN-Token* and *RNN-Token* can detect some zero-day attacks (*e.g.*, $D$-3 and $D$-7), but the performances are very unstable. The bad detection performance of these supervised approaches is mainly because of the training set. Since these approaches use WAF labels for training, the positive samples in the training set are those *WAF-malicious* requests which are dropped by WAF, which are likely to be quite different from the actual zero-day attacks.

Unsupervised baseline approaches *SAE*, *Hmmpayl* and *DFA* also perform badly. They achieve low precisions, recalls and F1-scores. In *SAE*, the dimensionality reduction of encoder misses important features from the token sequences, and makes it unsuitable to deal with long requests with complex patterns. *Hmmpayl* performs badly because it only works well when the number of states is equal to the length of sequences. However, it cannot process the requests well if it has many states [5]. Note that, we only have *Hmmpayl*'s results in some traces since they ran out of memory in others. *Hmmpayl* extracts sliding windows from sequences. It uses $(l-n+1)\times n$ for an $l$-length sequence ($n$ denotes window size), consuming almost $n$ times memory than the original sequence. *DFA* also performs badly. The various types of HTTP requests made the automata built too complex and the generalizing of the modeling reduces its detection accuracy since our data volume is far beyond the scenario in our its original design.

Above observations can be best explained by the zero-day attack example in Table V, which was detected by *ZeroWall*, *CNN-Token*, and *RNN-Token*, but not by other approaches. The attack hidden in the message body (in bold font) is an injection attack against PHP servers. This attack aims to collect more information from the server by forcing the server-side programs to execute malicious codes from the attacker. The injection detection rules used in WAF are usually based on keywords and regular expressions, *e.g.*, *eval*, *request*, *select* and *execute*. However, the attack in Table V contains none of these keywords, thus successfully cheated the WAF. On the other hand, *ZeroWall* is based on the "understanding" of benign requests without worrying *exactly how* the attacks deviate from benign patterns. The structure of above zero-day attack request is more like a programming language, which is obviously different from a normal request which usually transmits data, thus *ZeroWall* was able to successfully detect this attack. *CNN-Token* and *RNN-Token* were able to detect this specific zero-day attack because this attacks' tokens happen to overlap with some tokens in the training set, a snippet of which is shown in Table VI. However, not all zero-day attacks have overlapping tokens with known attacks, which explains the *CNN-Token* and *RNN-Token* worse performance than *ZeroWall*.

TABLE V: A Real Case of Zero-Day Attack

| |
|---|
| ...<br>searchword=d&**order=**}{**end if**}{**if:1)print_r($_POST[func]**<br>**($_POST[cmd]));//**}{**end if**}**&func=assert&cmd=phpinfo();** |
| Token Sequence: search php searchtype _pnum_0_ _OTHER_ onechr order end if if _pnum_1_ _OTHER_ _post _OTHER_ _post cmd end if _OTHER_ assert cmd phpinfo |

TABLE VI: Training set's token sequences overlapping with those of the zero-day attack in Table V. Highlighted in bold.

| | |
|---|---|
| **1** | plus ad_js php aid _pnum_0_ onechr **assert** _pnum_1_ execute execute function bd byval onechr for onechr _pnum_2_ to len onechr step _pnum_3+_ onechr mid onechr _pnum_3+_ if isnumeric mid onechr _pnum_3+_ then execute bd bd chr onechr else execute bd bd chr onechr mid onechr _pnum_3+_ onechr _pnum_3+_ **end if** chr _pnum_3+_ next end function response write execute on error resume next bd _phex_0_ response write response end |
| **2** | preview php _OTHER_ php **assert** _OTHER_ onechr |
| **3** | lib _OTHER_ module inc php _OTHER_ eval _OTHER_ onechr class _OTHER_ onechr **phpinfo** |
| **4** | cms _OTHER_ uploads _OTHER_ php id **assert** _OTHER_ eval base64_decode **_post** z0 z0 _pbas_0_ |
| **5** | myship php cmd eval base64_decode **_post** z0 z0 _pbas_0_ |

### *F. True Positives and WAF Rules*

Besides the example in Table V, *ZeroWall* detects many other zero-day Web attacks. Security engineers from company $I$ manually checked the results of *ZeroWall* and found 141583 true zero-day attacks. These zero-day attacks can be divided into 28 categories, including webshell, SQL injection, probing, trojan and other exploiting against particular applications. For each category, the security engineers have already composed a new WAF rule to detect these attacks in the future.

### *G. False Positives and Whitelist*

Table VII uses trace $D$-7's statistics to show the effect of *ZeroWall*'s whitelist mechanism. In this experiment the whitelist is updated daily. The numbers of whitelist rules refer to how many whitelist rules are added each day, based on the FPs labeled *on that day*. For example, 16 FPs were found during manual investigation on 0602's results. Based on these FPs, security engineers composed 5 whitelist rules and added them. These 5 rules were applied to the 0603's data and reduced 0603's FPs by 70 (222-152). The results shows that the whitelist reduces the number of FPs with low overhead (numbers of rules are very small). Based on these results, we believe *ZeroWall* is practical in real-world deployment.

TABLE VII: Results of whitelist. *No WL* and *WL* means the results without and with whitelist, respectively. No whitelist is applied on 0602 because it is the first day during testing.

| Date | Precision | | # of FP | | # of white-list rules |
|---|---|---|---|---|---|
| | No WL | WL | No WL | WL | |
| 0602 | 0.9972 | - | 16 | - | 5 |
| 0603 | 0.9643 | 0.9753 | 222 | 152 | 3 |
| 0604 | 0.9580 | 0.9999 | 310 | 1 | 1 |
| 0605 | 0.9731 | 0.9944 | 320 | 65 | 6 |
| 0606 | 0.9845 | 0.9993 | 121 | 5 | 1 |
| 0607 | 0.9672 | 1.0000 | 194 | 0 | 0 |

TABLE VIII: Training and testing speed with and without hash table (requests/s). The incoming requests refer to the average number of requests received by the customer per second.

| Trace | Incoming Requests | Training | | Testing | |
|---|---|---|---|---|---|
| | | No Hash | Hash | No Hash | Hash |
| *D*-1 | 2.60 | 1.09 | 256.89 | 229.24 | 39634.40 |
| *D*-2 | 5.19 | 3.72 | 202.13 | 785.75 | 65556.80 |
| *D*-3 | 22.44 | 7.09 | 835.43 | 514.33 | 50420.17 |
| *D*-4 | 25.83 | 5.42 | 1014.67 | 305.42 | 50913.24 |
| *D*-5 | 52.45 | 12.48 | 1046.55 | 414.86 | 38132.88 |
| *D*-6 | 294.30 | 1.47 | 4001.95 | 70.04 | 176255.90 |
| *D*-7 | 873.36 | 3.23 | 4262.48 | 53.77 | 88989.06 |
| *D*-8 | 883.16 | 6.67 | 6389.23 | 142.29 | 106692.90 |

### *H. ZeroWall's Training and Testing Overhead*

We evaluate the training and detection speed using *the number of requests processed per second*, on *one* server with the following configuration: Intel(R) Xeon(R) Gold 6148 CPU 2.40GHz $*$ 2, 512GB RAM. Note that in reality more servers can be used. The results in Table VIII show that the overhead of *ZeroWall* is low and our proposed hash table mechanism can reduce the overhead by 2 to 3 orders of magnitude.

## VI. Discussions

**Similarity Metrics:** We now justify BLEU as our similarity metric and demonstrate how to empirically determine the anomaly detection thresholds. Fig. 6 plots the CDFs for distributions of malicious scores of benign requests and zero-day attacks, where X-axis denotes the value of malicious score and Y-axis represents the cumulative distribution. Obvious differences can be noticed between BLEU_BENIGN and BLEU_ZERODAY curves: over 90% of the benign requests have malicious scores smaller than 0.3, while the malicious scores of the BLEU_ZERODAY are mostly distributed larger than 0.95, evidenced by the sharp increase in the CDF of BLEU_ZERODAY 0.95. One empirical method for setting the threshold is to automatically scan the thresholds and find the one with most dividing power (*e.g.*, red vertical line in Fig. 6).

Fig. 6 also justifies our selection of BLEU as the similarity metrics out of several popular metrics (BLEU, GLEU [31], NIST [40], CHRF [41], [42]) in neural machine translation field, because the differences between the benign and zero-day BLEU curves is stably (across different $D$-$n$'s) larger than those alternative metrics. GLEU has slightly worse performance, followed by NIST, while CHRF is the worst.

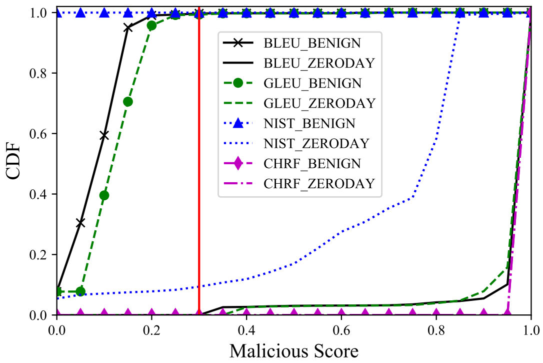

Fig. 6: CDF curves for malicious score distribution of benign requests and zero-day attacks with different similarity metrics.

**Limitations:** One limitation of our unsupervised approach is that too small volume of data penalizes the performance. As a neural network based approach, the volume of the training data is very important. And a small dataset may contain very rare (or none) zero-day attacks, which makes it hard to evaluate our approach. Fortunately, we have shown that *D*-1, with weekly trace size of 1.6 million requests, works pretty well already.

**Evasions:** A fundamental assumption of this approach is that most of the historical logs are benign. Intuitively, this approach may look like a good target for poisoning attacks. During poisoning attacks, the attacker would inject large amount of malicious samples in the normal traffic, hoping the system would learn from the wrong dataset. However, poisoning attacks have few impact on *ZeroWall* because: *ZeroWall* is deployed behind WAF, which means the poisoning samples are filtered by the WAF first before read by *ZeroWall*. It is unrealistic for an attacker to inject enough malicious samples.

## VII. Conclusion

This paper advocates the general framework of augmenting existing signature-based WAFs with unsupervised machine learning based zero-day Web detection approach. We believe this approach is immediately deployable and widely-applicable in real-world. To the best of our knowledge, this paper is the first to formulate the zero-day Web attack detection problem into a neural machine translation quality assessment problem, a direction along which we believe more solutions can be invented. We implements the above idea by adopting a state-of-art neural machine translation algorithms, i.e., encoder-decoder recurrent neural networks. In our evaluation using large-scale real-world traces, *ZeroWall* achieves high F1-scores over 0.98, significantly outperforming existing approaches.

## Acknowledgment

Our study was supported by the National Key R&D Program of China (Grand No.2019YFB1802504), National Natural Science Foundation of China (Grant 61572278 & U1736209) and Beijing National Research Center for Information Science and Technology (BNRist). We are also very thankful for all those valuable comments given by anonymous reviewers.

# REFERENCES

[1] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM, 2003.

[2] Dorothy E Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987.

[3] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[4] Yingbo Song, Angelos D Keromytis, and Salvatore J Stolfo. Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic. In *NDSS*, volume 9, pages 1–15. Citeseer, 2009.

[5] Davide Ariu, Roberto Tronci, and Giorgio Giacinto. Hmmpayl: An intrusion detection system based on hidden markov models. *computers & security*, 30(4):221–241, 2011.

[6] Nidhi Srivastav and Rama Krishna Challa. Novel intrusion detection system integrating layered framework with neural network. In *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, pages 682–689. IEEE, 2013.

[7] Ming Zhang, Boyi Xu, Shuai Bai, Shuaibing Lu, and Zhechao Lin. A deep learning method to detect web attacks using a specially designed cnn. In *International Conference on Neural Information Processing*, pages 828–836. Springer, 2017.

[8] Ali Moradi Vartouni, Saeed Sedighian Kashi, and Mohammad Teshnehlab. An anomaly detection method to detect web attacks using stacked auto-encoder. In *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, pages 131–134. IEEE, 2018.

[9] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[10] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298. ACM, 2017.

[11] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. Tiresias: Predicting security events through deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 592–605. ACM, 2018.

[12] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *Quality of Service (IWQoS), 2017 IEEE/ACM 25th International Symposium on*, pages 1–10. IEEE, 2017.

[13] Ke Wang, Gabriela Cretu, and Salvatore J Stolfo. Anomalous payload-based worm detection and signature generation. In *International Workshop on Recent Advances in Intrusion Detection*, pages 227–246. Springer, 2005.

[14] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.

[15] Namit Gupta, Abakash Saikia, and D Sanghi. Web application firewall. *Indian Institute of Technology, Kanpur*, 61:62, 2007.

[16] Michael Becher. *Web application firewalls*. VDM Verlag, 2007.

[17] Dennis Appelt, Cu D Nguyen, and Lionel Briand. Behind an application firewall, are we safe from sql injection attacks? In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, pages 1–10. IEEE, 2015.

[18] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chavez. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[19] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[20] Leyla Bilge and Tudor Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 833–844. ACM, 2012.

[21] Amazon web services. Website. https://en.wikipedia.org/wiki/Amazon_Web_Services.

[22] Alibaba cloud. Website. https://en.wikipedia.org/wiki/Alibaba_Cloud.

[23] F5 networks. Website. https://en.wikipedia.org/wiki/F5_Networks.

[24] Reyadh Shaker Naoum, Namh Abdula Abid, and Zainab Namh Al-Sultani. An enhanced resilient backpropagation artificial neural network for intrusion detection system. *International Journal of Computer Science and Network Security (IJCSNS)*, 12(3):11, 2012.

[25] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. Long short term memory recurrent neural network classifier for intrusion detection. In *Platform Technology and Service (PlatCon), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[26] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. In *ICLR 2018*, 2018.

[27] Malek Ben Salem and Salvatore J Stolfo. Detecting masqueraders: A comparison of one-class bag-of-words user behavior modeling techniques. *JoWUA*, 1(1):3–13, 2010.

[28] Kenneth L Ingham, Anil Somayaji, John Burge, and Stephanie Forrest. Learning dfa representations of http for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.

[29] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[30] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 187–196. International World Wide Web Conferences Steering Committee, 2018.

[31] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[32] Neural machine translation (seq2seq) tutorial. http://www.dnhys.com/.

[33] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[34] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR 2015*, 2015.

[35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[36] Deborah Coughlin. Correlating automated and human assessments of machine translation quality. In *Proceedings of MT summit IX*, pages 63–70, 2003.

[37] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

[38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145. Morgan Kaufmann Publishers Inc., 2002.

[41] Maja Popović. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, 2015.

[42] Maja Popović. chrf deconstructed: beta parameters and n-gram weights. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, volume 2, pages 499–504, 2016.