Anomaly-Based Web Attack Detection: A Deep Learning Approach

Jingxi Liang
Peking University
Beijing, China100871
jingxil@pku.edu.cn

Wen Zhao
Peking University
Beijing, China100871
zhaowen@pku.edu.cn

Wei Ye
Peking University
Beijing, China100871
wye@pku.edu.cn

ABSTRACT

As the era of cloud technology arises, more and more people are beginning to migrate their applications and personal data to the cloud. This makes web-based applications an attractive target for cyber-attacks. As a result, web-based applications now need more protections than ever. However, current anomaly-based web attack detection approaches face the difficulties like unsatisfying accuracy and lack of generalization. And the rule-based web attack detection can hardly fight unknown attacks and is relatively easy to bypass. Therefore, we propose a novel deep learning approach to detect anomalous requests. Our approach is to first train two Recurrent Neural Networks (RNNs) with the complicated recurrent unit (LSTM unit or GRU unit) to learn the normal request patterns using only normal requests unsupervisedly and then supervisedly train a neural network classifier which takes the output of RNNs as the input to discriminate between anomalous and normal requests. We tested our model on two datasets and the results showed that our model was competitive with the state-of-the-art. Our approach frees us from feature selection. Also to the best of our knowledge, this is the first time that the RNN is applied on anomaly-based web attack detection systems.

CCS Concepts

• Security and privacy→ Intrusion detection systems; • Computing methodologies→Neural networks; • Security and privacy→Web protocol security.

Keywords

web security, HTTP requests, anomaly detection, deep learning, recurrent neural network

1. INTRODUCTION

Web applications play an important role in people's daily life especially when people are starting to move their applications and personal data to the cloud. The prevalence of web applications and the large amounts of private user data being stored in them make themselves attractive attack targets. Hence protecting web applications from intrusions is vital. Among all kinds of vulnerabilities, HTTP request related vulnerabilities which are exploited by sending deliberately designed requests consists the largest portion. According to [1], injections such as SQL injection and Cross-Site Scripting are listed as the first and third most critical web application security risks.

Generally, there are two approaches to detect attacks mentioned above. The first is the signature-based method, which is to look for specific attack patterns in requests; The second is anomaly-based which is to establish normal request profiles so that anomalous requests can be discriminated from normal ones. The signature-based method is adopted more wildly than the anomaly-based method because usually the signature-based one has lower false alarm rate and achieves higher accuracy. For example, ModSecurity, one popular open source Web Application Firewall (WAF), builds up OWASP ModSecurity Core Rule Set (CRS) containing a massive number of rules which can detect SQL Injection, Cross Site Scripting, HTTP Protocol Violations and etc. As effective as it is, the rule-based method is still problematic. Firstly, it is only as good as the extent of the rule set, which means it is incapable of identifying attacks that are not in its signature dataset. Secondly, bypassing WAF can be done easily by replacing keywords of existing malicious requests or encoding themselves multiple times [2][3]. Thirdly, extremely large attack pattern set or requests with long lengths consumes lots of computing resources to finish pattern comparing.

In this paper we present a novel anomaly detection approach utilizing the RNN with the Long-Short Term Memory (LSTM)unit or Gated Recurrent Unit(GRU)unit. Our model takes Uniform Resource Locators (URLs) in the HTTP GET requests as the input. After the URLs are tokenized by following a certain strategy which reduces the variability while preserves the intrinsic information, two RNNs that have learned the normal request patterns output their "familiarities" with given URLs. And then a trained neural network decides whether given requests are anomalous based on the output of the RNNs. Our model is self-learning and free of feature selection. It can be customized to learn normal request patterns for any specific web applications.

The rest of the paper is structured as follows: Section 2 summarizes recent related works on anomaly detection of web applications. In Section 3 we introduce the motivation. In Section 4 we present the architecture of our model. In Section 5 we discuss the experiments in detail. Finally, we draw our conclusions in Section 6.

2. RELATED WORKS

Kruegel et al. [4] associated separated models with different features of a request such as query attribute length, attribute character distribution, attribute order and query attribute presence

ICNCC 2017, December 8–10, 2017, Kunming, China

ACM ISBN 978-1-4503-5366-3/17/12…$15.00
DOI: https://doi.org/10.1145/3171592.3171594

and etc. Each model outputs a probability value for each attribute of a request which reflects the likelihood of the occurrence of the given feature value with regards to an established profile. The whole query is marked as anomalous if one or more features' probabilities exceed the determined threshold. The anomaly detection system proposed by Kirchner et al. [5] also used several classifiers as feature analyzers. However, K-nearest neighbor classification was adopted in Kirchner's approach, allowing for comparison of every request to be classified with all normal requests stored in the pool feature by feature.

Ingham et al. [6] modeled token sequences parsed from normal requests using the Deterministic Finite Automata (DFA) in combination with rules. Malicious web requests are detected by determining similarities between a request and the DFA. Furthermore, Rieck et al. [7] proposed an anomaly detection method based on language model such as n-grams and a similarity measurement between n-gram sequences.

Duc Le [8] explored the possibility of detecting anomaly via the Self Organizing Map (SOM) approach. This approach is believed to have the ability to form well-separated clusters for normal and abnormal requests according to the assumption of various kinds of requests having distinct behaviors.

In domains other than web application security, the RNN is wildly employed to deal with time series problems. For example, Malhotra et al. [9] leveraged the LSTM-RNN on ECG, space shuttle, power demand, and multi-sensor engine datasets to detect anomalies. Unfortunately, web application anomaly detection systems based on RNNs are not available yet, to the best of our knowledge. Our model treats every URL as a token sequence in time order which is similar to [6] and [7]. But a new strategy is chosen to split the URL. Also, the RNN is used in a different way compared with those in other fields. Instead of predicting the next value and computing differences between the ground-truth value and predicted value as in [9], our model computes its "familiarity" with the token after being provided with all preceding tokens in the same token sequence.

3. MOTIVATION

In GET requests almost all anomalies are related to the URLs. A URL, also termed a web address as well, usually is composed of one absolute path and several query parameters. It always conforms to the syntax as follows [10]:

"http:" "//" host [":" port] [abs_path ["?" query]]

where abs_path is used to locate the resource and optional query string is used to pass parameters to the resource which always follows a question mark.

We believe useful information that can help detect anomalies is stored in the order and values of URL sequence, from the perspective of treating a URL as a time series. For example, id=1/*union*/union/*select*/select+1,2,3/*is definitely an attack because a query parameter value consisting of special characters and words should not be appended after the equal sign, given that the value of id attribute usually is a numerical value. Also provided that email=who@me.com is normal, email=1 is considerably an anomaly because a numerical value does not comply with the format of email. A deliberately modified absolute path such as visiting hidden resources can lead to a static web attack, and delicately designed values of the query attribute suck as SQL Injection can cause a dynamic web attack. Given that web applications would generate almost all links for users to click, there is no reason for them to manually type a long URL. Hence changes on query parameter order and occasional absence of query values could be regarded as illegitimate.

The principles in detecting anomalies in other types of requests like POST requests are similar. But extra efforts to handle different request formats are required. Hence we only focus on URLs of HTTP GET requests.

4. ARCHITECTURE

Our model first trains two RNNs using only tokenized normal URLs. Formally, training set of URLs is denoted as $U = \{u_1, u_2 \cdots u_n\}$ where u_i represents the i_{th} URL.And $y_i \in \{0,1\}$indicates$u_i$is an anomalous request when $y_i = 1$. After the tokenizing procedure,

$$\vec{w}^{ups} := [w_1^{ups}, w_2^{ups}, w_3^{ups} \cdots]$$

$$\vec{w}^{qps} := [[w_{11}^{qps}, w_{12}^{qps}, w_{13}^{qps} \cdots], [w_{21}^{qps}, w_{22}^{qps}, w_{23}^{qps} \cdots], \cdots]$$

are obtained where thew_i denotes the i_{th} token in the sequence. And structural information of the URL path is stored in $\vec{w}^{ups}$while structural information of the query parameters is in $\vec{w}^{qps}$within which each row represents one query parameter.

To exploit continuous real-vector representation, each token w is mapped to a learned distributed feature vector $\overrightarrow{em_w} \in R^M$before being fed into the RNNs. This guarantees the ability not only to generalize compared with one-hot encoding but also to decrease the dimensions needed to represent a token.

Separately, two RNNs are trained using normal requests so that the RNNs can "familiarize" with legitimate request patterns. One RNN is dedicated to the URL path structure patterns, and the other one concentrates on query parameter structure patterns. The training process can be analogized as asking the RNN to pick out the token which it thinks is most likely to occur next, with all prior tokens in the token sequence allowed to be seen. It can be described as the model taking $\overrightarrow{em_w}_1, \overrightarrow{em_w}_2, \cdots, \overrightarrow{em_w}_{i-1}$as the input and attempting to predict w_i. The model will be penalized if it makes a wrong choice. After training, the RNN is able to give the occurrence probability of the succeeding token as long as it is provided with all previous tokens in the sequence. Well-trained RNNs mark the end of the first phase.

In the second phase, a neural network classifier is trained on $(\overrightarrow{prob}_i, y_i)$ pairs to learn whether one URL is normal where constant-sized vector$\overrightarrow{prob}$is from $\overrightarrow{prob}^{ups}$ and $\overrightarrow{prob}^{qps}$.

Our model starts to report web anomalies once training process is completed and all learned parameters are fixed. The overall architecture of our model is summarized in Figure 1. Sections below discuss our model in detail.

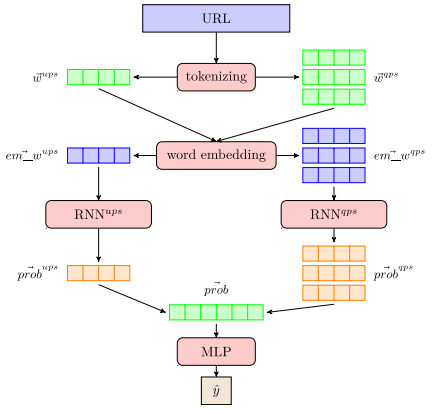

Figure 1. Overall architecture

4.1 URL TOKENIZING

Apparently attacks or anomalies can be categorized into two types according to the place that does not conform to the normal patterns. The first type of anomalies has the abnormal query parameter structure which dynamic web attacks and wrong data formats belong to. The second type of anomalies has the anomalous URL path structure which static web attacks belong to. Hence two cases need to be taken into consideration. For URL path structure the authorized choices of path name and query attribute keys are limited, and the order between them are fixed. And for query parameter structure the possible query values are infinite. But luckily, although the variability of query values is high, they have to obey certain format rules. The URL tokenizing procedure is designed to capture the essential information in terms of URL path structure and query parameter structure and to reduce the variability. The general tokenizing procedure includes:

- Decode URL and map all upper case letters in URL to lower case letters.
- Obtain the token sequence of URL structure by parsing URL following certain rules and substitute query values with <VAL> token.
- Obtain the token sequence for each query parameter in a URL by splitting query parameters of the URL by non-word characters, and replacing numerical values with <NV> token and string values with <SV> token in query values.
- Insert <START> token at the beginning of and append <END> token at the end of each sequence.
- Construct a vocabulary using training data with the least k%frequent tokens dropped.
- Look up every token of sequences in vocabulary built before and substitute tokens which are not in vocabulary with <UNK> token.

Replacing the least k% frequent tokens with <UNK> token enables our model to deal with out-of-vocabulary tokens. For instance, in the URLs that link to static resources like /path/to/cat.jpg the name of static resources might change constantly. It is possible that the filenames are not included in training set. Without the <UNK> token, the model has no idea what to do with unexpected tokens. Figures 2 gives an example about how a URL is tokenized. Because our model is designed to protect a single web application in which the host name and the port number will not change, the host and port parts are abandoned during URL tokenizing.

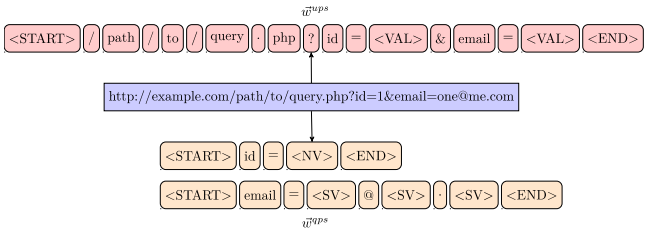

Figure 2. One example about how a URL is tokenized

4.2 WORD EMBEDDING

The benefits of word embedding were mentioned in [11]. It is a concept from natural language processing where words are mapped to vectors of real numbers. Compared with one-hot encoding which uses one dimension for each word in the vocabulary, word embedding represents word in a continuous vector space with much lower dimension. Additionally, words with semantic similarities tend to be nearby in the vector space of word embedding after training. In the context of web anomaly detection, this can be applied as tokens appearing in a similar structure are more likely to have a closer proximity. Our model could benefit from those two aspects. A look-up table for tokens is constructed where the i_{th} row represents the embedding vector $\overrightarrow{em\_w_i} \in R^D$ of w_i. Formally, it can be written as below:

$$\overrightarrow{em\_w} = \vec{w}\vec{M}_{look\_up}$$

where $\vec{w} \in R^V$ is an one-hot vector derived from token w, $M_{look\_up} \in R^{V*D}$ is the lookup table. V represents the vocabulary size and D is the feature size of the feature vector.

4.3 RECURRENT NEURAL NETWORK

The RNN is a type of artificial neural networks which are applicable to sequential inputs. It is supposed to be able to connect things happened previously to the present task by storing environment information in an inner state. Additionally, since they are recurrent, they can handle arbitrary length of sequences easily as well as output a value at each timestep. In our model the RNN is used to output the occurrence probability of the token at every timestep. In other words, the RNN is used to memorize the patterns of normal URLs.The input at timestampt is defined as $\vec{x}_t := [\overrightarrow{em\_w_1}, \overrightarrow{em\_w_2} \cdots \overrightarrow{em\_w_t}]$ and the label or the excepted value is $y_t := w_{t+1}$. Let $h_t^l \in R^H$ be a hidden state in layer l at time t in our stacked multilayer RNN model where H is the dimension of hidden states. Then h_t^l is calculated as below:

$$\vec{h}_t^1 = \text{rnn}(\overrightarrow{em\_w_t}, \vec{h}_{t-1}^1)$$

$$\vec{h}_t^l = \text{rnn}(\vec{h}_t^{l-1}, \vec{h}_{t-1}^l) 1 < l \leq L$$

where L is the number of RNN layers. Figure 3 roughly explains the structure of the RNN we used. And the rnn function in vanilla RNNs isusually:

$$\text{rnn}(\vec{h}_t^{l-1}, \vec{h}_{t-1}^l) := \tanh(\vec{h}_t^{l-1}W + \vec{h}_{t-1}^l U)$$

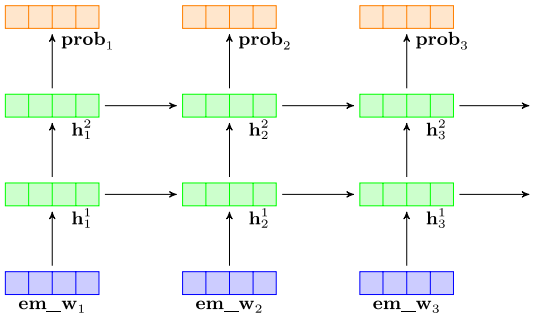

Figure 3. Structure of unfolded recurrent neural network

The output of the RNN at every timestep is expected to be a vector of which the length is the vocabulary size so that each value in the vector represents the occurrence probability. But the output of the RNN network $\vec{h}_t^L$ is in R^H. To bridge this gap a projection matrix $W^{proj} \in R^{H*V}$ is constructed. The $\overrightarrow{prob}_i$ is defined as:

$$\overrightarrow{logits}_t = \vec{h}_t^L W^{proj}$$

$$\overrightarrow{prob}_t^i = \frac{\exp(\overrightarrow{logits}_t^i)}{\sum_{j=1}^V \exp(\overrightarrow{logits}_t^j)}$$

Using vanilla RNNs might face the difficulty called the vanishing gradient problem [12] during training. This problem can be solved by using delicate unit design such as the LSTM structure or the GRU structure. According to [13] these gated RNNs like the

LSTM-RNN and GRU-RNN are better than vanilla RNNs in performance.

4.4 LONG-SHORT TERN MEMORY

The Long-Short Term Memory is proposed by Hochreiter et al.[14] in 1997 to make the RNN learn to store information over extended time intervals more quickly. Here, we adopt the LSTM variant similar to [15] but without peep-hole connections. The LSTM-RNN has the ability to remember values for either long or short duration of time as indicated in its name. In our case considering a long URL /path/z…/y?b=2&h=2&z=1 where the occurrence of last query key is only dependent on the second part of the path, it is hard for vanilla RNN to learn this association quickly due to every timestep's information counts in vanilla RNNs. But for the RNN with the LSTM structure it is much easier since it can choose to forget information in between. At timestep tthe inputs of LSTM cell are the hidden state of LSTM cell from beneath layer $\vec{h}_t^{l-1}$, the hidden state of LSTM cell from previous timestep$\vec{h}_{t-1}^{l}$ and so called cell state of LSTM cell from previous timestep $\vec{C}_{t-1}^{l}$. First we compute cell information $\vec{\tilde{C}}_t^{l}$ which is going to be used to update the cell state. Then a forget gate $\vec{f}_t^{l}$used to decide what information is thrown away from $\vec{C}_{t-1}^{l}$is computed. Next, an input gate $\vec{i}_t^{l}$ used to decide which part of information is useful in $\vec{\tilde{C}}_t^{l}$is calculated. Afterward current LSTM cell state $\vec{C}_t^{l}$ is computed under the help of $\vec{C}_{t-1}^{l}$, $\vec{f}_t^{l}$ and $\vec{i}_t^{l}$. Subsequently an output gate $\vec{o}_t^{l}$is computed to determine what information is going to output. Finally, the output of LSTM cell $\vec{h}_t^{l}$ is obtained using $\vec{o}_t^{l}$ and $\vec{C}_t^{l}$. The formal procedure is as follows:

$$\vec{\tilde{C}}_t^{l} = \tanh\left(\vec{h}_t^{l-1}W_C^{l} + \vec{h}_{t-1}^{l}U_C^{l} + \vec{b}_C^{l}\right)$$
$$\vec{f}_t^{l} = \sigma\left(\vec{h}_t^{l-1}W_f^{l} + \vec{h}_{t-1}^{l}U_f^{l} + \vec{b}_f^{l}\right)$$
$$\vec{i}_t^{l} = \sigma\left(\vec{h}_t^{l-1}W_i^{l} + \vec{h}_{t-1}^{l}U_i^{l} + \vec{b}_i^{l}\right)$$
$$\vec{C}_t^{l} = \vec{f}_t^{l} \oplus \vec{C}_{t-1}^{l} + \vec{i}_t^{l} \oplus \vec{\tilde{C}}_t^{l}$$
$$\vec{o}_t^{l} = \sigma\left(\vec{h}_t^{l-1}W_o^{l} + \vec{h}_{t-1}^{l}U_o^{l} + \vec{b}_o^{l}\right)$$
$$\vec{\mathrm{h}}_{\mathrm{t}}^{\mathrm{l}} = \vec{o}_t^{l} \oplus \vec{C}_t^{l}$$

where $\oplus$ denotes element wise operations and sigmoid function σ is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ and hyperbolic tangent function tanh is defined as $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$. Here, W_*^*, U_*^* and $\vec{b}_*^*$ are trainable parameters.

4.5 GATED RECURRENT UNIT

The Gated Recurrent Unit was introduced by Cho et al. [16] recently to capture dependencies over various time intervals. It is similar to the LSTM but has a simpler structure. Compared with the LSTM, the GRU does not have a separate cell state as well as the output gate. Instead the GRU uses the update gate $\vec{z}_t^{l}$ and reset gate $\vec{r}_t^{l}$the former of which functions like combination of the input and forget gates in the LSTM and the latter of which is like deciding what information should be used to make up candidate activation. Similarly, the inputs of GRU cell at timestep t are the activation of GRU cell from beneath layer $\vec{h}_t^{l-1}$ and the activation of GRU cell from previous timestep $\vec{h}_{t-1}^{l}$. The $\vec{\tilde{\mathrm{h}}}_{\mathrm{t}}^{\mathrm{l}}$and $\vec{\mathrm{h}}_{\mathrm{t}}^{\mathrm{l}}$ are the candidate activation and activation respectively while $W_*^*, U_*^*$and $\vec{b}_*^*$ are parameters to learn. The formal definition is as below:

$$\vec{z}_t^{l} = \sigma\left(\vec{h}_t^{l-1}W_z^{l} + \vec{h}_{t-1}^{l}U_z^{l} + \vec{b}_z^{l}\right)$$
$$\vec{r}_t^{l} = \sigma\left(\vec{h}_t^{l-1}W_r^{l} + \vec{h}_{t-1}^{l}U_r^{l} + \vec{b}_r^{l}\right)$$
$$\vec{\tilde{\mathrm{h}}}_{\mathrm{t}}^{\mathrm{l}} = \tanh\left(\vec{h}_t^{l-1}W_h^{l} + (\vec{r}_t^{l} \oplus \vec{h}_{t-1}^{l})U_h^{l}\right)$$
$$\vec{\mathrm{h}}_{\mathrm{t}}^{\mathrm{l}} = \left(1 - \vec{z}_t^{l}\right) \oplus \vec{h}_{t-1}^{l} + \vec{z}_t^{l} \oplus \vec{\tilde{\mathrm{h}}}_{\mathrm{t}}^{\mathrm{l}}$$

4.6 MULTILAYER PERCEPTRON

Given the probability sequences of URLs calculated by our RNNs, it is still not straightforward for some URLs to determine whether it is normal. Hence the Multilayer Perceptron (MLP) is adopted. The MLP is a feedforward artificial neural network. It is supposed to learn how to distinguish occurrence probability sequences of normal and anomalous URLs.

It supervisedly learns a function $\hat{\mathrm{y}}_{\mathrm{i}} = \mathrm{f}(\vec{\mathrm{prob}}_{\mathrm{i}}): \mathrm{R}^{\mathrm{d}} \rightarrow \mathrm{R}^{0}$ where $\hat{\mathrm{y}}_{\mathrm{i}} \in \{\mathrm{normal}, \mathrm{abnormal}\}$ and $\vec{\mathrm{prob}}_{\mathrm{i}}$ is the fixed length vector occurrence probability sequences derived from $\vec{\mathrm{prob}}^{ups}$ and $\vec{\mathrm{prob}}^{pps}$. The process of our K layers MLP is as follows:

$$\vec{\mathrm{output}}_1 = \max\left(0, \vec{\mathrm{prob}}W_1 + \vec{b}_1\right)$$
$$\vec{\mathrm{output}}_{\mathrm{k}} = \max\left(0, \vec{\mathrm{output}}_{\mathrm{k-1}}W_k + \vec{b}_k\right)\ 1 < k \leq K$$
$$\hat{\mathrm{y}}_{\mathrm{i}} = \mathrm{argmax}\left(\vec{\mathrm{output}}_{\mathrm{K}}W_o + \vec{b}_o\right)$$

where W_*represents the weight matrix and $\vec{b}_*$ represents the bias vector. The max function is the activation function which provides the non-linearity and the argmax function is to obtain the index of the maximum value.

5. EXPERIMENTS

Our model was tested on two datasets. One is a public available dataset developed by Carmen et al. [17] at the "Information Security Institute" of Spanish Research National Council (CSIC), and the other is a dataset consisting of extracted URLs from WAF log files. Our model will be trained and tuned on the training set and test set is only used when assessing the performance of our model.

5.1 Datasets

The first dataset we used is HTTP dataset CSIC [17] which contains generated HTTP traffic targeted to an E-commerce website. There are 36,000 normal requests and more than 25,000 anomalous requests in this dataset. All anomalous requests are categorized into static attacks which try to visit hidden (or non-existent) resources, dynamic attacks which include SQL injection, CRLF injection, cross-site scripting, buffer overflows and unintentional illegal requests such as not having the same structure as normal parameter values. Since our focus is on URLs of GET Requests, we extract the URLs of GET requests from the CSIC dataset and remove duplicated URLs to form a new dataset. Also the new dataset is randomly split into a training set and a test set. The trimmed dataset from CSIC is described in Table 1:

Table 1. Trimmed HTTP DATASET CSIC 2010

Type		Training Set	Test Set	Total
Normal		7464	1866	9330
Anomalous	Static Attacks	941	246	1187
	Dynamic Attacks	2651	662	3313
	Illegal Requests	2900	715	3615

Another dataset is generated from HTTP traffic recorded by the WAF. In order to get a larger dataset without duplicated items, URLs from different domains are merged by unifying domain names, resource paths and attribute keys. As a result, a total of

2500 anomalous and 2500 normal unique URLs are collected among which all of anomalies are dynamic attacks. The dataset is also separated into training set and test set as in Table 2 for model validating purposes.

Table 2. Dataset from the WAF logs

Type	Training Set	Test Set	Total
Normal	2000	500	2500
Anomalous	2000	500	2500

5.2 PERFORMANCE MEASUREMENTRS

The performance of models is measured in terms of accuracy, sensitivity and specificity. The accuracy measurement shows the model's capability to correctly classify data. But in security field a high false positive rate is intolerable, as a very small rate of false positives can quickly render an intrusion detection system unusable [18]. Hence the sensitivity measurement is used to test the model's ability to detect anomalies from all anomalous data and the specificity denotes the percentage of accurately detected normal data among all normal data.

$$\text{Accuracy} = \frac{\text{number of data classified correctly}}{\text{Total number of data}}$$

$$\text{Sensitivity} = \frac{\text{number of anomalies classified correctly}}{\text{Total number of anomalies}}$$

$$\text{Specificity} = \frac{\text{number ofnormal data classified correctly}}{\text{Total number of normaldata}}$$

5.3 RESULTS

We tested our model ten times and the mean of results on each dataset were used. Afterwards we compared our model with the WAF which is ModSecurity 2.7.7 powered by CRS Version 2.2.8 embedded in Apache web server as well as other approaches on CSIC Dataset. We only compared our model without RNN, our model with the GRU and our model with LSTM on WAF logs dataset.

The results on the CSIC dataset are described in Table 3 which includes classification performances of SOM, C4.5, Naive Bayes, X-means and EM approaches evaluated in[8][1]. It turned out our model outperformed all other models on the CSIC dataset in terms of accuracy. In other measurements our model was also competitive. The WAF reported almost zero false negative rate on the CSIC dataset which is stunning. However, the sensitivity of WAF was far too low. The reasons for this low sensitivity are: 1) the WAF can barely detect anomalies like visiting non-existing resources, changing keys or order of query parameters and wrong formats of query parameter values; 2) dynamic attacks can find their way to bypass WAF by changing keywords or encoding themselves multiple times. The C4.5 reported the highest sensitivity while performed poor in terms of specificity. Our model with LSTM achieved the best result in accuracy and also showed quite good sensitivity and specificity which are 97.56% and 99.21% respectively.

The RNN module of our model has proved itself important to our model. On both datasets our model exceled the one without RNN module. Between the GRU-RNN and the LSTM-RNN it is hard to decide which is better because the performance is so close. On the CSIC dataset our model with the LSTM-RNN defeated the one with the GRU-RNN. And the LSTM-RNN won by 0.54% in accuracy, 0.34% in sensitivity and 0.76% in specificity. But on the WAF logs dataset the opposite condition occurred which was that our model with the GRU-RNN outperformed the one with the LSTM-RNN by 0.18% in accuracy, 0.22% in sensitivity and 0.14% in specificity.

Table 3. Comparison on CSIC dataset

Method	Accuracy	Sensitivity	Specificity
ModSecurity with CRS	0.5520	0.0436	**0.9941**
EM	0.7486	0.7516	0.7478
X-means	0.7493	0.6837	0.9865
Na ïve Bayes	0.8408	0.5235	0.9286
SOM	0.9282	0.9497	0.9242
C4.5	0.9650	**0.9914**	0.8697
Our model without RNN	0.8515	0.7403	0.9546
Our model with GRU	0.9788	0.9722	0.9845
Our model with LSTM	**0.9842**	0.9756	0.9921

Table 4. Comparison on WAF logs dataset

Method	Accuracy	Sensitivity	Specificity
Our model without RNN	0.9475	0.9462	0.9488
Our model with GRU	**0.9856**	**0.9880**	**0.9832**
Our model with LSTM	0.9838	0.9858	0.9818

5.4 DISCUSSIONS

Our model achieved quite a good result, which was above 98% accuracy in both datasets and no measurement was below 97.5%. It is truly beyond our expectations. In the WAF's defense we did not deploy the commercial rules and for the convenience of testing we only enabled SQL Injection, XSS and several other types of rule files which we think are related with the attack types in the datasets.

Interestingly, if we quantify our model's familiarity toward each token in URL sequences by the negative logarithm of corresponding probability computed by our model, we can obtain Figure 4 and 5. In every normal URL, the familiarity value of each token is relatively small while there is at least one quite large familiarity value in anomalous URLs which forms the pike in Figure 4 and 5. It showed our model could memory the normal URL patterns.

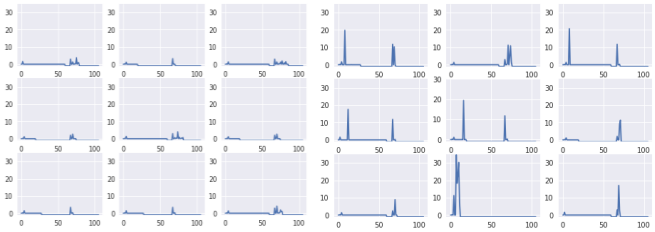

Figure 4. RNN's familiarities toward 9 normal URLs (Left) and 9 anomalous URLs (Right) in CSIC test dataset. All samples are randomly picked. The x-axis which ranges from 0 to 105 represents the index of familiarity sequence $\vec{s}$ and the y-axis which ranges from -1 to 35 represents the value of the familiarity metric calculated by the corresponding RNN. The values equal to -1 indicate padded positions in the token sequence.

[1] Performances on HTTP GET data of models implemented by us are not as good as those in [8]. Hence, we used their results.

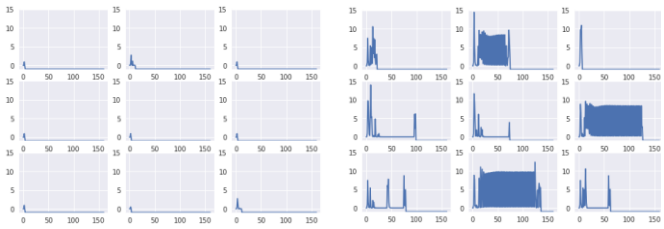

Figure 5. RNN's familiarities toward 9 normal URLs (Left) and 9 anomalous URLs (Right) in WAF logs test dataset. All samples are randomly picked. The x-axis which ranges from 0 to 160 represents the index of familiarity sequence $\vec{s}$ and the y-axis which ranges from -1 to 15 represents the value of the familiarity metric calculated by the corresponding RNN. The values equal to -1 indicate padded positions in the token sequence.

Our model also has its limitations. First of all, after inspecting the misclassified URLs, it seems our model cannot handle some kinds of long URLs very well. Secondly, we cannot come up with a method to dynamically leverage our model between the true positive rate and the false positive rate after deploying on the condition that we want to keep the RNN + MLP architecture. For example, other models often have a threshold value, and by modifying the threshold value dynamically they can sacrifice the specificity to enhance the sensitivity at runtime.

6. CONCLUSIONS

In this work we proposed a novel anomaly detection approach for web applications which leveraged the RNN with a delicate cell structure such as the LSTM and GRU to learn patterns of normal requests, which then used a MLP classifier to predict whether a request is normal or anomalous based on the outputs of the RNNs mentioned above. Our model was tested on two datasets. The results showed that the performance of our model was competitive with the state-of-the-art.

7. ACKNOWLEDGMENTS

This work is fully supported by National Engineering Research Center for Software Engineering of Peking University.

8. REFERENCES

[1] TopOWASP.2017.Top10-2017.*TheTenMostCriticalWebApplicationSecurity Risks* (2017).

[2] OWASP. 2017. SQL Injection Bypassing WAF. (2017). Retrieved May 6, 2017 fromhttps://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF

[3] PavolLupták. 2011. Bypassing Web Application Firewalls. In *Proceedings of 6th International Scientific Conference on Security and Protection of Information*. 79–88.

[4] Christopher Kruegel and Giovanni Vigna. 2003. Anomaly detection of web- based attacks. In *Proceedings of the 10th ACM conference on Computer and communications* security. ACM, 251–261.9

[5] Michael Kirchner. 2010. A framework for detecting anomalies in http traffic using instance-based learning and k-nearest neighbor classification. In *Security and Communication Networks (IWSCN), 2010 2nd International Workshop on*. IEEE, 1–8.

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

[7] Konrad Rieck and Pavel Laskov. 2006. Detecting unknown network attacks using language models. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 74–90.

[8] Duc Le Jr. 2017. *An Unsupervised Learning Approach for Network and System Analysis*. Master'sThesis. Dalhousie University

[9] PankajMalhotra,LovekeshVig,GautamShroff,andPuneetAgarwal.2015.Long short term memory networks for anomaly detection in time series. In *Proceedings*. Presses universitaires de Louvain, 89.

[10] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. 1999. *Hypertext transfer protocol–HTTP/1.1*. Technical Report.

[11] YoshuaBengio,RéjeanDucharme,PascalVincent,andChristian Jauvin.2003.A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[12] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6, 02 (1998), 107–116.

[13] Junyoung Chung, CaglarGulcehre, KyungHyun Cho, and YoshuaBengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[14] SeppHochreiterandJürgenSchmidhuber.1997.Longshort-termmemory.*Neural computation* 9, 8 (1997), 1735–1780.

[15] Graves Alex. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[16] Kyunghyun Cho, Bart Van Merriënboer, DzmitryBahdanau, and YoshuaBengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[17] CT Gimnez, Alejandro Pérez Villegas, and GA Maranón. 2010. HTTP data set CSIC 2010. (2010).

[18] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 305–316.