ComEX

LETTER

Evaluation of HTTP request anomaly detection model using fastText and convolutional autoencoder

Haruta Yamada[1, a)] **and Ryoichi Kawahara**[1, b)]

Abstract With the advent of the Internet and its close connection to people's lives, web applications have become increasingly important. To ensure that the web application is secure, a web application firewall (WAF) detects and stops attacks that exploit application vulnerabilities in communication with server applications. However, these firewalls require continuous tuning by experts with in-depth knowledge of the technologies and services provided, which may become a major obstacle to the introduction of WAF. To resolve this problem, we developed two autoencoder-based models based on an unsupervised learning model that uses only normal requests, considering the implementation and operation costs. We then evaluated the performance of the two autoencoder-based models. The first model converts a hypertext transfer protocol (HTTP) request into ASCII codes and learns their relationship in a normal request using an autoencoder. The second model generates an array of word vectors using fastText and learns using a convolutional autoencoder, which solves the problem identified in the performance evaluation of the first model where the problem was that the simple conversion to ASCII codes was not enough to distinguish between normal and anomalous requests. The two models were evaluated using the HTTP DATASET CSIC2010 dataset. The AUC for the second model was approximately 0.94 while that for the first model was approximately 0.71. This means that the second model has higher accuracy despite being an unsupervised approach, one that does not require labeled anomalous requests, and can be applied with low costs.

Keywords: web application firewall, anomaly detection, autoencoder, fastText
Classification: Network

1. Introduction

With the advent of the Internet and its close connection to people's lives, web and mobile applications have become commonplace. In such a scenario where private user information may be vulnerable to unauthorized access, server application security is one of the most important aspects of deployment. In the "10 Major Information Security Threats 2023" published by the Information-technology Promotion Agency, Japan, threats related to server applications are ranked in both individual and organizational sections. Attack techniques that exploit web application vulnerabilities are also diverse. Examples include structured query language (SQL) injection, cross-site scripting, and operating system (OS) command ejection.

Web application security is ensured by a web application firewall (WAF), which detects and blocks attacks that exploit server vulnerabilities during communication (mainly hypertext transfer protocol (HTTP)/hypertext transfer protocol secure (HTTPS) requests) with server applications.

Recently, research has been conducted on WAF that uses machine learning. There are two main methods. The first method uses a supervised machine learning model with labeled normal and anomalous requests (i.e., each request is labeled as normal or anomalous in the training data), whereas the second method uses an unsupervised machine learning model with only normal requests. Jemal et al. [1] developed a convolutional neural network-based supervised learning model. The model converts an HTTP request into ASCII codes when it is input. This method of converting a request to ASCII codes increases the amount of information in the input data and affects the model's processing time; hence, only parameters that affect the accuracy of the model are extracted from the request. Mac et al. [2] developed an unsupervised learning model using an autoencoder and extended autoencoder. Before inputting the data into the model, this method replaces the parameters with different tokens to enable the model to interpret the parameters according to certain rules. The proposed regularized deep autoencoder achieves high accuracy comparable to that of supervised learning models. Yan et al. [3] developed an unsupervised anomaly detection algorithm based on self-translation machine with attention mechanism. Similar to Mac et al. [2], they adopted the approach of replacing the parameters with different tokens to enable the model to interpret them.

Supervised learning models must use appropriately labeled normal/anomalous requests to cover various patterns of attacks to minimize the risk of unknown attacks. Compared with supervised learning models, unsupervised learning models can be developed and deployed at relatively lower operation costs because they only require the collection of normal requests. However, because only normal requests are used, defining the boundary between normal and anomalous requests is difficult.

In this study, we developed a machine learning model with an emphasis on unsupervised learning taking into account the operation costs, and evaluated its performance. The goal was to develop a model that is accurate as a pure classifier and that considers implementation and operation costs. To ensure that the model can interpret the parameters, it is necessary to replace certain parameters with the tokens, as used in the previous studies. However, selecting parameters to be placed for applications that change daily can be challenging. Furthermore, the requirement to develop a dedicated parser

[1] Faculty of Information Networking for Innovation and Design, Toyo University, Kita-ku, Tokyo 115-8650, Japan
a) s1f101900273@iniad.org
b) ryoichi.kawahara@iniad.org

DOI: 10.23919/comex.2024XBL0060
Received March 21, 2024
Accepted April 8, 2024
Publicized May 9, 2024
Copyedited July 1, 2024

may result in significant implementation costs. Therefore, we did not adopt the token substitution approach; instead, we focused on developing a model using only a simple replacement process. Specifically, an approach of conversion to ASCII codes or conversion to a word vector called fastText [4] was used.

Many examples of using autoencoders have been reported in studies on anomaly detection tasks [5, 6, 7], where the autoencoders are a type of neural network consisting of an encoder and a decoder for dimensionality compression and restoration, respectively. Therefore, this study developed a model based on an autoencoder.

In this study, we developed and evaluated the performance of two autoencoder-based anomaly detection models. In particular, the first method uses ASCII code conversion, whereas the second method uses fastText and a convolutional autoencoder.

2. First method with ASCII code

2.1 CSIC2010

Before introducing the first method, we describe the dataset used in this study. We evaluated the performance of the two methods through numerical experiments with the HTTP DATASET CSIC2010 [8] (CSIC2010). CSIC2010 is a dataset that has been used in several WAF studies, such as those mentioned in related studies, and it includes requests for an e-commerce web application. This dataset was published by the Spanish Research National Council. The dataset is divided into training and test datasets. The training dataset comprises 36,000 normal requests, whereas the test dataset comprises 36,000 normal and approximately 25,000 anomalous requests, respectively.

2.2 Preprocessing of input data

We adopted an approach from [1] that converts an HTTP request into ASCII codes and passes them to the input layer of an autoencoder-based model. Using ASCII codes increases the amount of information and affects processing time; hence, only the attack-specific parameters (method, URL, and content) were extracted from the request and input into this model. Specifically, the following procedure was used to convert the requests:

1. Concatenate each parameter of the request into a single string.
2. Convert each character of the string concatenated in step 1 into an ASCII code (numeric value).
3. Divide the converted ASCII code by 128 and obtain the remainder.
4. Divide the remainder by 128 to obtain a number between 0 and 1. Then, send it to the input layer.

2.3 Model construction

The model used in the first method has the following structure: The input and output layers comprised 1197 neurons that matched the longest string in the training and test datasets used in this study. The middle layer consisted of five layers, with 128, 64, 32, 64, and 128 neurons in each layer. The rectified linear unit (ReLU) and sigmoid functions were the activation functions for the middle and output layers, respectively. The mean-squared error (MSE) was used as the loss function. This model contained 328,525 parameters.

2.4 Evaluation

Several methods that use unsupervised learning models have been proposed for classification tasks. In this study, the MSE values of the input and output data were used. Specifically, after the model was trained using a training dataset of normal requests, a request in a test dataset was tested; the request that outputs an MSE (i.e., MSE for the individual elements of the input/output vector for the request) above a preset threshold was classified as anomalous (this type of anomaly detection is based on [5], for example). In our evaluation, the threshold was set to be x percentile of MSEs of individual requests of the training dataset, where $x = 90$ is an example as shown later.

The frequency distributions of the MSEs for the training dataset, normal and anomalous data in the test dataset are shown in Figs. 1, 2 and 3, respectively. The position of each percentile (80, 85, 90, and 95) of the training dataset is shown in each graph as a reference.

Figures 1 and 2 show that most normal requests have negligible MSEs; however, groups were formed between

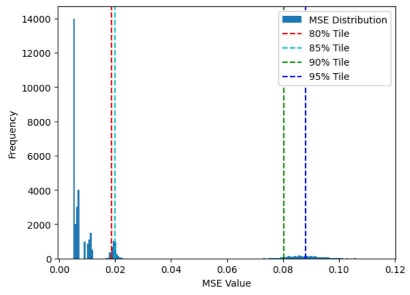


Fig. 1 MSE distribution of the training dataset.

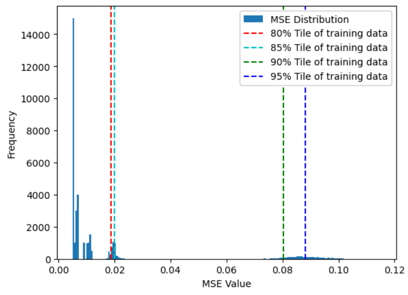


Fig. 2 MSE distribution of the normal data from the test dataset.

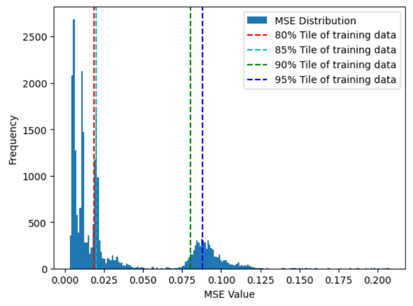
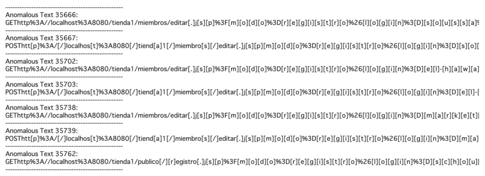


Fig. 3 MSE distribution of the anomalous data from the test dataset.



```
------------------------------------------------
Anomalous Text 35666:
GEThttp%3A//localhost%3A8080/tienda1/miembros/editar[.]j[s][p]%3F[m][o][d][o]%3D[r][e][g][i][s][t][r][o]%26[l][o][g][i][n]%3[D][s][o][u][s][s][a]
------------------------------------------------
Anomalous Text 35667:
POSThtt[p]%3A/[/]localhos[t]%3A8080[/]tiend[a]1[/]miembro[s][/]editar[.]j[s][p][m][o][d][o]%3D[r][e][g][i][s][t][r][o]%26[l][o][g][i][n]%3[D][s][o]
------------------------------------------------
Anomalous Text 35702:
GEThttp%3A//localhost%3A8080/tienda1/miembros/editar[.]j[s][p]%3F[m][o][d][o]%3D[r][e][g][i][s][t][r][o]%26[l][o][g][i][n]%3[D][e][l]-[h][a][w][a]
------------------------------------------------
Anomalous Text 35703:
POSThtt[p]%3A/[/]localhos[t]%3A8080[/]tiend[a]1[/]miembro[s][/]editar[.]j[s][p][m][o][d][o]%3D[r][e][g][i][s][t][r][o]%26[l][o][g][i][n]%3[D][e][l]-
------------------------------------------------
Anomalous Text 35738:
GEThttp%3A//localhost%3A8080/tienda1/miembros/editar[.]j[s][p]%3F[m][o][d][o]%3D[r][e][g][i][s][t][r][o]%26[l][o][g][i][n]%3[D][m][a][r][k][e][t]
------------------------------------------------
Anomalous Text 35739:
POSThtt[p]%3A/[/]localhos[t]%3A8080[/]tiend[a]1[/]miembro[s][/]editar[.]j[s][p][m][o][d][o]%3D[r][e][g][i][s][t][r][o]%26[l][o][g][i][n]%3[D][m][a]
------------------------------------------------
Anomalous Text 35762:
GEThttp%3A//localhost%3A8080/tienda1/publico[/][r]egistro[.]j[s][p]%3F[m][o][d][o]%3D[r][e][g][i][s][t][r][o]%26[l][o][g][i][n]%3[D][s][c][h][o][u]
------------------------------------------------
```

Fig. 4 Normal requests with particularly high MSEs.

0.08 and 0.10. Figure 3 shows that the MSEs for anomalous requests are marginally higher than those for normal data; however, over half of the data points are classified as normal, even when the 80 percentile of the training dataset is used as the threshold.

Figure 4 shows a list of normal requests with particularly high MSEs. The character codes where differences higher than the threshold are detected are highlighted with [], indicating that they are particularly concentrated in the query parameters and contents. That is, the MSEs of normal requests with numerous query parameters or contents increased because character codes were learned individually rather than as a single coherent character string (word).

3. Second method with fastText and convolutional autoencoder

This method uses the following three techniques to improve the performance of the first method.

1. Generate word vectors to capture word features.
2. Exclude features (protocol, host name, etc.) that are common to normal and anomalous requests to emphasize features more.
3. Introduce a convolution layer to capture the surrounding word features.

Although there are several methods for generating word vectors, we chose fastText, which is an open-source library published by the Facebook AI Research Laboratory. It is particularly used for tasks involving text representation and classification. We applied it to our method to generate word vectors from requests each of which is broken down into words. This is because, one of the fastText's features is the support for subwords, which is expected to enable us to deal with unknown words and/or synonyms specific to anomalous data.

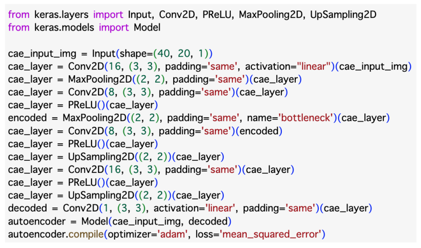
3.1 Preprocessing of input data

The procedure for processing the conversion of a request is as follows:

1. Extract specific parameters (path, query parameters, request body, etc.) from the request.
2. Divide each parameter into words.
3. The segmented words are combined into a single array[1].
4. Generate a word vector array using fastText pre-trained using the training dataset.

In this study, method, path, query parameters, and content were selected as the parameters to be extracted. Note that, since some parameters may have a significant impact on the performance of the model, care should be taken in selecting parameters when developing models for different applications.

[1] The example is as follows: ['GET', 'tienda1', 'publico', 'autenticar.jsp', 'modo', 'entrar', 'login', 'aguistin', 'pwd', 'iNnota', 'remember', 'on', 'B1', 'Entrar'].



```python
from keras.layers import Input, Conv2D, PReLU, MaxPooling2D, UpSampling2D
from keras.models import Model

cae_input_img = Input(shape=(40, 20, 1))
cae_layer = Conv2D(16, (3, 3), padding='same', activation="linear")(cae_input_img)
cae_layer = MaxPooling2D((2, 2), padding='same')(cae_layer)
cae_layer = Conv2D(8, (3, 3), padding='same')(cae_layer)
cae_layer = PReLU()(cae_layer)
encoded = MaxPooling2D((2, 2), padding='same', name='bottleneck')(cae_layer)
cae_layer = Conv2D(8, (3, 3), padding='same')(encoded)
cae_layer = PReLU()(cae_layer)
cae_layer = UpSampling2D((2, 2))(cae_layer)
cae_layer = Conv2D(16, (3, 3), padding='same')(cae_layer)
cae_layer = PReLU()(cae_layer)
cae_layer = UpSampling2D((2, 2))(cae_layer)
decoded = Conv2D(1, (3, 3), activation='linear', padding='same')(cae_layer)
autoencoder = Model(cae_input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

Fig. 5 Convolutional autoencoder.

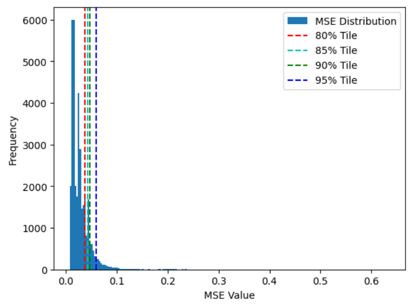


Fig. 6 MSE distribution of the training dataset under second method.

3.2 Model construction

We used a convolutional autoencoder to capture the word features surrounding the word vector array. The word vector size was set to 20 dimensions, and the number of words that could be input simultaneously (number of words per request) was set to 40, which is larger than the maximum for the training and test datasets. For the activation function of the convolution layer, we used the parametric ReLU (PReLU) function instead of the ReLU function because the word vectors generated by fastText may take negative values and because we input directly into the model without normalization. The Python code for the model is shown in Fig. 5[2]. The number of parameters was reduced to 8,417, which is approximately 1/40 that of the first method.

3.3 Evaluation

The evaluation was performed using the same dataset as in the first method. The MSE frequency distributions for each dataset are shown in Figs. 6, 7, and 8. Compared with the first method, the MSEs increased slightly for normal data, and the 80 percentile increased from approximately 0.02 to 0.035. However, the leftward shift of the independent peaks of the graph observed in the first method resulted in the 95 percentile decreasing from approximately 0.09 to 0.05. Furthermore, MSEs significantly increased for anomalous data. The number of data points that were classified as normal despite being anomalous by each threshold value was significantly reduced.

Finally, we compared the receiver operating characteristic (ROC) curves for the first and second methods by plotting the

[2] We performed zero padding on the input array to reach the specified dimension size when the number of words was insufficient. (We also did something similar in the first method.)

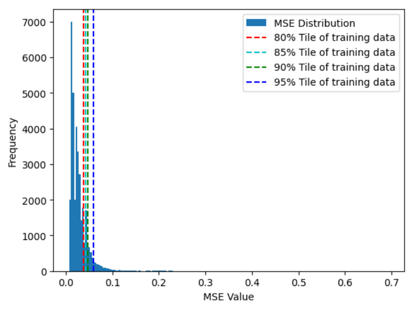


Fig. 7 MSE distribution of the normal data from the test dataset under second method.

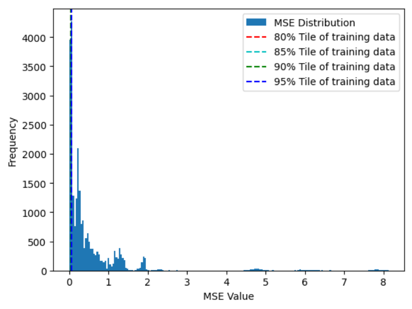


Fig. 8 MSE distribution of the anomalous data from the test dataset under second method.

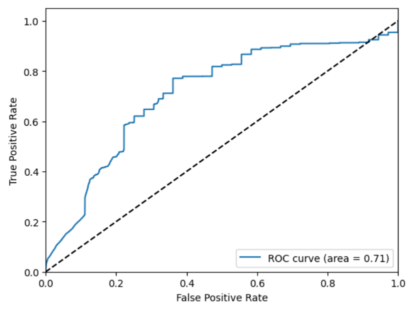


Fig. 9 ROC curve and AUC score for first method.

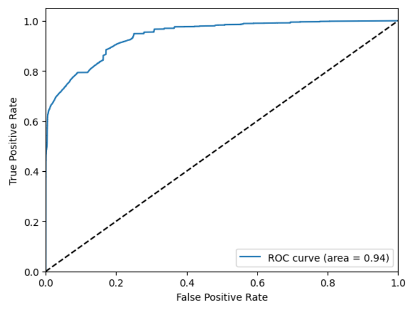


Fig. 10 ROC curve and AUC score for second method.

curves using normal and anomalous data of the test dataset (Figs. 9 and 10). The results showed that the area under the ROC curve (AUC) scores for the first and second methods were approximately 0.71 and 0.94, respectively, confirming an improvement in the score. For reference, when the threshold was set to the 90 percentile under the second method, the precision and recall scores were approximately 0.846 and 0.854, respectively.

4. Conclusion

In this study, to develop a simple unsupervised-learning-based anomaly detection method for WAFs, we developed two autoencoder-based models that consider implementation and operation costs. Instead of adopting the token substitution approach used in related studies, the first method used a simple approach in which an HTTP request is first converted into ASCII codes and then the autoencoder learns the relationship of a normal request. Based on the evaluation results of the first method, we proposed a second method that uses fastText and a convolutional autoencoder to solve the problems experienced by the first method. We confirmed that the second method detects anomalies more accurately than the first method because it generates smaller and larger errors for normal and anomalous requests, respectively. This means that the second model has higher accuracy despite being an unsupervised approach, one that does not require labeled anomalous requests, and can be applied with low costs.

Although the model developed in the evaluation using the CSIC2010 dataset showed a certain level of accuracy, it is necessary to confirm whether the model has the same accuracy for services with different characteristics. Therefore, further research is required.

Acknowledgments

We would like to thank Editage (https://www.editage.jp/) for English language editing.

References

[1] I. Jemal, M.A. Haddar, O. Cheikhrouhou, and A. Mahfoudhi, “SWAF: A smart web application firewall based on convolutional neural network,” 2022 15th International Conference on Security of Information and Networks (SIN), Nov. 2022. DOI: 10.1109/SIN56466.2022.9970545

[2] H. Mac, D. Truong, L. Nguyen, H. Nguyen, H.A. Tran, and D. Tran, “Detecting attacks on web applications using autoencoder,” Proc. 9th Int. Symp. Inf. Commun. Technol., pp. 416–421, Dec. 2018. DOI: 10.1145/3287921.3287946

[3] L. Yan and J. Xiong, “Web-APT-detect: A framework for web-based advanced persistent threat detection using self-translation machine with attention,” *Lett. IEEE Comput. Soc.*, vol. 3, no. 2, pp. 66–69, 2020. DOI: 10.1109/LOCS.2020.2998185

[4] https://github.com/facebookresearch/fastText, last accessed March 2024.

[5] Y. Ikeda, K. Ishibashi, Y. Nakano, K. Watanabe, and R. Kawahara, “Anomaly detection and interpretation using multimodal autoencoder and sparse optimization,” arXiv:1812.07136v1, Dec. 2018.

[6] X. She and Y. Sekiya, “A convolutional autoencoder based method for cyber intrusion detection,” IEICE Tech. Rep., vol. 120, no. 414, IN2020-77, pp. 138–143, March 2021.

[7] N. Ogawa and R. Kawahara, “Network anomaly detection and failure scale estimation method,” IEICE Tech. Rep., vol. 123, no. 177, NS2023-57, pp. 32–37, Sept. 2023 (in Japanese).

[8] HTTP DATASET CSIC2010, Spanish Research National Council, https://www.tic.itefi.csic.es/dataset/, accessed Nov. 2023.