Detection of Cross-Site Scripting Vulnerabilities using a Deep Learning Approach

Ahmed Al-Ajeli

***Abstract*—In this paper, a method is proposed to detect Cross-Site Scripting (XSS) vulnerabilities in website source code. The method employs deep learning techniques, specifically Convolutional Neural Networks (CNNs), for detection. First, the source code (HTML, CSS, and JavaScript) is converted into an image. This image is then fed into the CNN to extract features. The extracted features are subsequently passed to a fully connected neural network that classifies the input code as either vulnerable or benign. To evaluate the performance of the proposed method, a dataset containing both malicious and benign scripts is used. Experimental results demonstrate that the method achieves a promising performance compared to existing approaches.**

***Index Terms*—Cross-Site Scripting, vulnerability detection, deep learning, convolutional neural network, gray image.**

I. Introduction

Internet services such as online banking, medical care, e-commerce, social networking and others are rapidly growing, providing more opportunities to cybercriminals to exploit the systems providing these services. Internet services have become an essential part of our daily life, but users of these services are unaware of hackers' methods to steal their sensitive information. Spreading awareness among users about hacker behaviour is not a realistic solution to such a problem. Thus, the best way to protect users' information and their privacy is via developing automatic and robust methods to detect vulnerabilities (weaknesses) in web systems to improve their security aspects. Most web vulnerabilities result from incorrect practices by programmers during the development of the source code. The Open Worldwide Application Security Project (OWASP) is a nonprofit organization aiming to improve software security by issuing a report listing the top 10 vulnerabilities. This report is of great interest to many public and private organizations, as well as to individuals (see Table IV). It can be observed from the table that Cross-Site Scripting (XSS) represents one of the most prominent vulnerabilities on the list, rising from seventh position in 2017 to third position in 2021. This type of vulnerability has caused organizations and individuals to suffer financial losses amounting to several trillions of dollars. Therefore, many researchers have focused on developing robust and effective methods to identify and mitigate such vulnerabilities [1]–[3].

Several studies have been conducted to detect XSS vulnerabilities in web applications [4]–[8]. These studies represent traditional methods to address the problem of XSS attack detection. However, one limitation of these studies is that they cannot handle all types of XSS attacks. Moreover, to adapt to a new XSS type, these methods require major modifications. To avoid the limitations of traditional approaches, researchers have proposed alternative approaches [9], [10]. These approaches are based on machine learning techniques which offer several advantages over traditional methods. Machine learning-based approaches can capture general patterns across various XSS attack types and significantly increase the detection accuracy.

The XSS approaches adopting machine learning techniques can be grouped into three categories: supervised, unsupervised and reinforcement learning [11]. In the first category, the dataset provided to the learning algorithm is assumed to be labeled. Several techniques have been developed in this category. In [12], they address the problem using the linear support vector machine; they achieved 95.4% detection accuracy. Another method which focuses on the feature extraction part has been proposed [13]. As a final step, the proposed method provides the extracted features to a set of machine learning techniques such as the decision tree and Naïve Bayes to make the final decision. The authors have reported that the decision tree algorithm achieved the highest detection accuracy. An example of this category is the work presented in [14], where the k-means clustering algorithm is applied. In addition, reinforcement learning models have been used in the context of XSS vulnerabilities detection [15], [16].

Although machine learning-based XSS attack detection approaches have become increasingly popular, they still face challenges in handling a variety of XSS attacks. Recent developments have introduced deep learning (DL) approaches [17] to address this problem. These approaches aim to improve the detection accuracy and cope with diverse XSS attacks. A plug-and-play ready-to-use device has been implemented for detecting XSS attacks, DOS attacks, and SQL attacks [18]. In this work, Convolutional Neural Network (CNN) [19] and Long Short-Term Memory networks (LSTM) models [20] have been adopted. Moreover, a multilayer perceptron (MLP) scheme which is integrated with the dynamic feature extractor for detecting XSS attacks has been proposed [21]. In addition, [22] has presented a strategy to detect one particular type of XSS attack, namely DOM-based XSS. Their approach combines DNN models and the taint tracking method for detecting such an attack. Recently, a new study combining the Universal Sentence Encoder (USE) with Word2Vec embeddings as a feature extractor has been introduced [23]. This approach led to improved performance of both machine learning and deep learning models.

The research in the present study is based on the concept of visualizing HTML scripts as grayscale images. It starts with a script and each character is converted into an integer code represented by 1 to 4 bytes. Then, all generated bytes are arranged into a two-dimensional array forming an image

Manuscript received April 11, 2025; revised August 6, 2025. This work was supported in part by the University of Babylon.

A. Al-Ajeli is the Head of Cyber Security Department at the College of Information Technology, the University of Babylon, Iraq (e-mail: a.alajeli@uobabylon.edu.iq).

TABLE I: OWASP Top 10 Vulnerabilities 2017–2021

Code	Vulnerability type (2017)	Vulnerability type (2021)
A01	Injection	Broken Access Control
A02	Broken Authentication	Sensitive Data Exposure
A03	Sensitive Data Exposure	Injection **(Cross-Site Scripting (XSS))**
A04	XML External Entities	Insecure Design **(new)**
A05	Broken Access Control	Security Misconfiguration
A06	Security Misconfiguration	Using Components with known Vulnerabilities
A07	Cross-Site Scripting	Broken Authentication
A08	Insecure Deserialization	Software and Data Integrity Failures **(new)**
A09	Using Components with Known Vulnerabilities	Insufficient Logging and Monitoring
A10	Insufficient Logging and Monitoring	Server-Side Request Forgery **(new)**

which is used to extract features for the classifier (detector). Images generated from malicious and benign scripts exhibit distinct textures, and the classifier is trained to recognize these texture patterns to distinguish between them. For feature extraction, a CNN model is applied, followed by a fully connected neural network with a single output neuron for binary classification.

The key contributions of this work can be outlined as:

- Visualizing the HTML script as grayscale images. Images representing malicious scripts exhibit consistent texture patterns, while those representing benign scripts display a distinct pattern.
- Adopting a deep learning approach that uses a CNN for feature extraction and a fully connected neural network as the classifier. The model takes as input the images generated during the visualization step and outputs the class label.
- Evaluating the performance of the proposed method on a dataset and comparing it with traditional methods to demonstrate its effectiveness.
- Achieving superior performance over existing methods with an accuracy of 99%, indicating the success of the proposed approach.
- Developing a model that is less complex than other methods used for detecting XSS vulnerabilities.

This paper is organized as follows. Section II gives a background on the XSS attacks and their main types in addition to deep learning and the CNN. A formal description of the problem being addressed is introduced in Section III. The proposed method and the approach to tackling this problem are covered in Section IV. Also, the experimental results obtained using the proposed method are presented in Section V, in addition to a discussion that justifies these results. This paper ends with a conclusion.

II. Background

A. Cross-Site Scripting

Cross-site scripting, also known as XSS, is one of the most popular attacks in which code (HTML tags or scripts) is injected into a target website. Such an attack worldwide has affected about 80% of web applications [11]. Any website is vulnerable to an XSS attack when accepting content from an untrusted source and inserting it into a benign website, unless proper input encoding or input validation is carried out. For example, if a malicious JavaScript code is inserted into the input fields (comments or any other fields) that receive some input from the user, and the malicious code is further executed by the browser, this implies that the website is vulnerable to an XSS attack. When the attackers run that code, they can impersonate the user on this website, i.e. they can perform operations allowed by the website on the user's behalf. Also, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site, and even rewrite the content of the HTML page. Three main types of XSS attack can be listed here [24]:

- **Reflected XSS:** This XSS attack occurs when a web application immediately returns a user input via a search result, error message or any other response including part or all of the input provided by the user as part of the request, without checking whether the input data is safe to render in the browser, and without permanent storing of the data provided by the user.
- **Stored XSS:** When a user input comes from a comment field, visitor log, a message forum, etc. and is stored in a database on the target server, it is generally said that a stored XSS attack has occurred. Consequently, a victim can retrieve the stored data from the database without considering if the data is safe to render in the browser.
- **DOM-Based XSS:** DOM-Based XSS is an XSS attack in which the DOM environment is modified leading to executing the attack payload in the victim's browser which executes the original client-side script, so that the client-side code runs unexpectedly. In other words, there does not exist a change in the page itself, but the client-side code included in the page runs differently as a result of the malicious modifications that occurred in the DOM environment.

B. Convolutional Neural Network

The convolutional neural network (CNN) is considered the most widely used deep learning model in learning features for large-scale image classification and recognition. Le Cun *at el* [25] originally proposed the CNN for a handwritten recognition task. Since then, many works have frequently appeared to report its success in numerous applications such as computer vision, speech recognition and natural language processing [26]. The CNN architecture has two main layers: the convolutional layer and the subsampling layer (also known as the pooling layer). The former layer implements the convolution operation to share weights and the latter layer reduces the dimensionality. Normally, the input of the network is a 2D array. For example, given an image x and a filter K (also called kernel), the convolution operation is defined as:

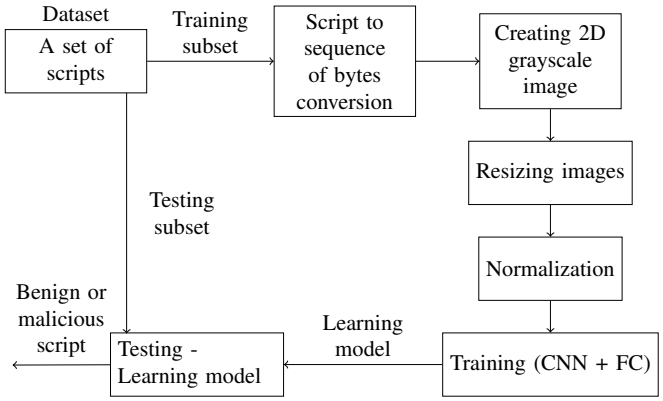


Fig. 1: The proposed approach

$$y_i = f(\sum_i K_{ij} \otimes x_i + b_j) \quad (1)$$

where y_i is the output of the convolution operation, b_j is the bias and f is a non-linear activation function. The convolution operation works by sliding windows of size 3×3 or 5×5 over the image to produce a feature map for each filter. Each element (also known as the receptive field) of the feature map represents the region in the input that produces the feature.

The dimensions of feature maps are reduced via the subsampling layer. This layer can typically be implemented by an average pooling operation or a max pooling operation. Multiple convolutional layers and pooling layers are stacked to allow a CNN to learn hierarchical feature representations of the input. The model learns the more abstract feature representation whenever using deeper layers. To predict a target (class), a fully connected neural network is added to the end of the CNN layers after implementing a flattening layer.

III. Problem Description

In this work, the problem of detecting XSS vulnerability in HTML files is reduced to a binary classification problem. Suppose there exists a set $X = \{x_1, \ldots, x_n\}$ of n HTML files each of which is associated with a label $y_i \in \{0, 1\}$ and $n = 1, \ldots, n$. The label $y = 1$ refers to the vulnerability of the file and $y = 0$ implies that the file is invulnerable. Each of these files has a different number of characters (string). The goal of the detection problem here is to build a model to predict y given X using the data being analyzed. Such a model can be expressed as a function $h : X \leftarrow \{0, 1\}$ which works as a classifier (detector). In other words, this is a supervised learning problem in which the model is trained until reaching the minimum error between the actual and estimated labels.

IV. Methods

In this section, the details of the present work will be covered. This work consists of four phases as shown in Fig. 1: 1) Starting with a dataset of scripts (HTML code embedding JavaScript code), a set of images is created. 2) Preprocessing is applied to the created images. 3) Then, the set of images is partitioned into two subsets: training and testing sets. The former is used to build the deep learning model, which will be used online to detect XSS vulnerabilities. 4) Finally, the result obtained by the learning model is evaluated using different criteria.

A. Dataset

The deep learning model was evaluated using a publicly available dataset from the Kaggle website. This dataset was compiled using XSS attack examples from cheatsheets provided by PortSwigger and OWASP. It contains both malicious (XSS) and benign scripts, organized in two columns: the *Sentence* column, which contains the script content, and the *Label* column, which indicates the class label (benign or XSS). Table II presents a snapshot of the dataset, showing row 0 (label 0, benign) and row 88 (label 1, XSS). The dataset comprises a total of $13,686$ samples, with $6,313$ benign scripts and $7,373$ XSS scripts.

B. Script to Image Conversion

This phase handles the conversion of the script, which is a string of characters, into a 2D image. This conversion takes two steps: (1) each character is encoded into 1-4 bytes (the number of bytes allocated to each character depends on the character itself). (2) The resulting sequence of bytes is visualized as follows: given a sequence S of k characters, this sequence is divided into n subsequences of m characters each. Then, a 2D $(n \times m)$ grayscale image is created (see Fig. 3). Consider that the last row of the image is padded with zeros when needed.

C. Preprocessing

CNN-based deep learning models require input images of a fixed size. However, since the script lengths vary, the resulting images also differ in size. To address this, all images are resized to a uniform dimension of 60×60 using the bicubic interpolation method [27]. The next preprocessing step involves normalizing the pixel values of the images to the range [0, 1] by dividing each grayscale value by 255.

D. The architecture of the CNN

The architectural details of the CNN model used here are described in this subsection. The architecture is summarized

Algorithm 1 Script to Image Conversion.

Input: S, a script of k characters.
Output: I, a 2D grayscale image of $n \times m$ dimensions.
1: Let L be an empty list.
2: **for** each $c \in S$ **do**
3: $e_c \leftarrow encode(c)$
4: Add e_c to L
5: **end for**
6: $n \leftarrow \lceil \frac{|L|}{m} \rceil$
7: $rem \leftarrow |L|$ mode m
8: pad $(m - rem)$ zeros to the end of L.
9: **while** L is not empty **do**
10: Move m elements from L to I.
11: **end while**

TABLE II: Snapshot of the Dataset

#	Sentence	Label
0	 Philosophy portal 	0
88	<style>@keyframes x{from {left:0;}to {left: 1000px;}}:target {animation:10s ease-in-out 0s 1 x;}</style>	1

in Table III, where the input is a 2D grayscale image representing the script. The model consists of three convolutional layers, each using a different number of kernels and the ReLU activation function. These convolutional layers are followed by max pooling layers, which serve to reduce the spatial dimensions of the feature maps generated by each convolution.

For example, the first convolutional layer takes an input of dimensions $60 \times 60 \times 1$ and transforms it into an output of $58 \times 58 \times 64$. This transformation is achieved by convolving the input image with 64 kernels of size (3×3), resulting in 64 feature maps. The output from this convolutional layer is then passed to a max pooling layer, where each 2×2 block in the feature maps is replaced with its maximum value. This reduces the output dimensions to $29 \times 29 \times 64$.

As shown in Table III, the output of the last max pooling layer is flattened (converted into a vector containing all feature maps) to prepare it for a series of fully connected layers of varying sizes. These layers form the final part of our deep learning model. The last layer is a sigmoid-activated output layer, which serves as the final classifier—i.e. the XSS detector. To train the model, binary cross-entropy is used as the loss function, which is standard for binary classification tasks like this. Additionally, the Adaptive Moment Estimation (Adam) algorithm is employed to optimize the loss function and minimize its value during training.

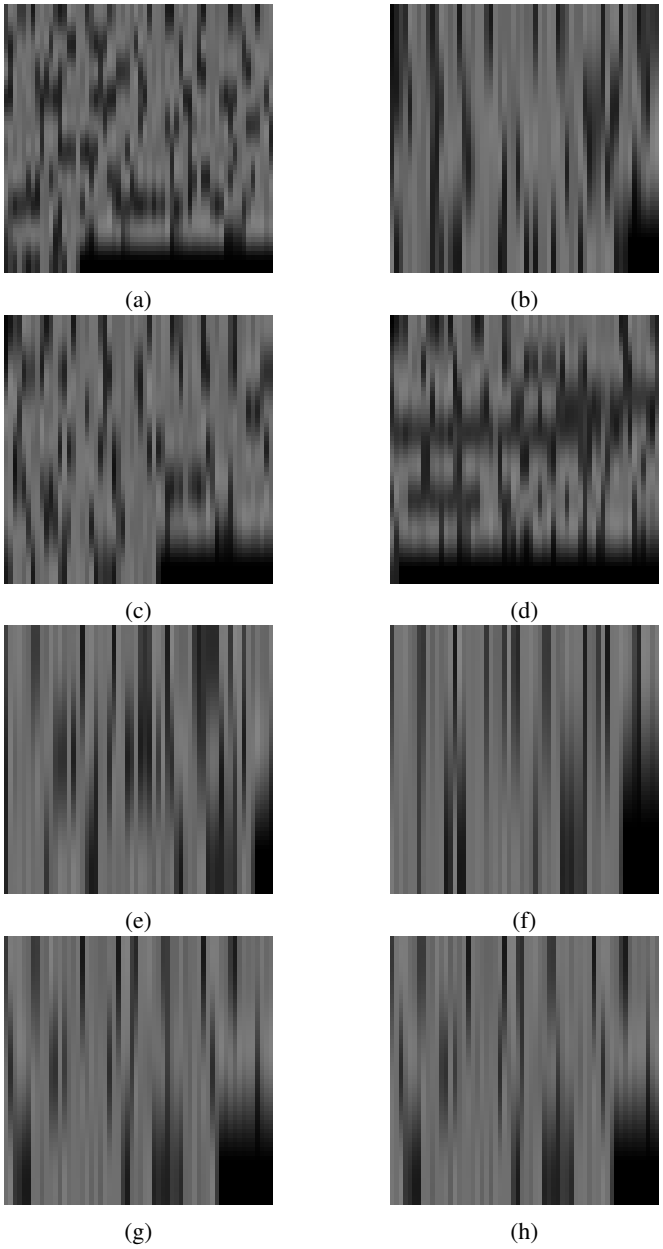


Fig. 2: Visual representation of scripts: (a)-(d) are benign scripts and (e)-(f) are XSS scripts

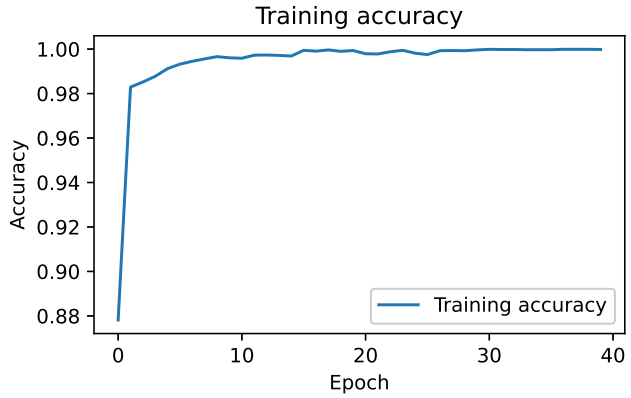
V. Results and Discussion

In this section, the experimental results obtained using the dataset described earlier are presented. The proposed method was implemented in Python using the TensorFlow/Keras library. Table IV outlines the hyperparameter settings used; these were selected through a trial-and-error process. A set of evaluation metrics - accuracy, recall, precision, F1-score and ROC was used to assess performance. Additionally, the confusion matrix was used to provide insights into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), offering a comprehensive view of the model's balance between detection and false alarm rates.

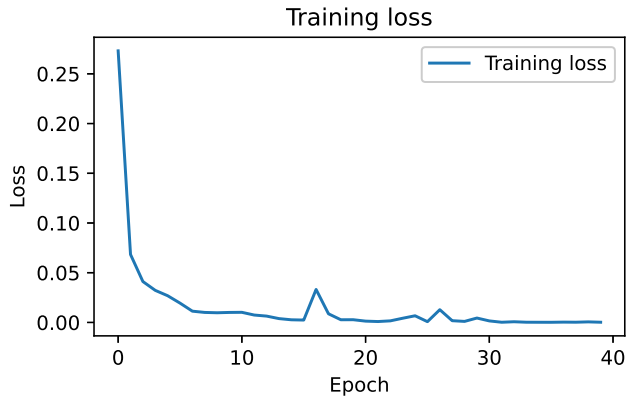
The first experiment investigated how the number of training iterations (epochs) impacts model accuracy and loss during training. As shown in Fig. 3, accuracy improves steadily with more epochs, starting at 0.73 in the first epoch and reaching 0.99 by the final epoch. Conversely, the loss

TABLE III: Deep Learning Model Architecture

Layer Number	Layer Type	Input Shape	Receptive Field	Number of Kernels
1	Convolution	$60 \times 60 \times 1$	$3 \times 3 \times stride\ 1$	64
2	Max Pooling	$58 \times 58 \times 64$	$2 \times 2 \times stride\ 2$	–
3	Convolution	$29 \times 29 \times 64$	$3 \times 3 \times stride\ 1$	128
4	Max Pooling	$27 \times 27 \times 128$	$2 \times 2 \times stride\ 2$	–
5	Convolution	$13 \times 13 \times 256$	$3 \times 3 \times stride\ 1$	256
6	Max Pooling	$11 \times 11 \times 256$	$2 \times 2 \times stride\ 2$	–
7	flatten	$5 \times 5 \times 256$	–	–
8	Fully Connected	6400	–	–
9	Fully Connected	256	–	–
10	Fully Connected	128	–	–
11	Sigmoid	64	–	–



(a) Accuracy increase during training



(b) Loss (error) decrease during training

Fig. 3: The performance of the training model

decreases from 0.48 to 0.006 over the same period. These results indicate significant accuracy gains and error reduction during training.

The other experiment evaluated the performance of the proposed approach using different baseline learning models such as Logistic Regression (LR), Support Vector Machine (SVM) and Random Forest (RF). Table V reports the results given by the baseline models and the present approach. It can be observed that the best result has been obtained when applying the proposed approach with respect to all evaluation metrics (indicated in bold). In addition, Table V compares the performance of the proposed approach with two state-of-the-art approaches reported in [23]. The first method utilizes the RF model, while the second employs a CNN-based deep learning approach. The proposed approach outperforms both, as indicated by the results in bold, achieving a superior accuracy of 0.9956. These result demonstrate that the proposed approach effectively captures the distinguishing visual features of benign and malicious scripts.

Despite only marginal improvements in evaluation metrics, the proposed model requires significantly fewer training iterations—from 100 epochs in previous studies to just 40. Moreover, the confusion matrix in Fig. 4 reveals that the proposed approach gives the lowest misclassification rates (with only 6 false negatives and 6 false positives). However, using the baseline models leads to higher false negatives and positives, for example the SVM model obtains 15 false negatives and 22 false positives. Only the reduction in false negatives is especially important, as it highlights the method's robustness in detecting actual threats. This indicates that the proposed method not only achieves high accuracy but also minimizes misclassification.

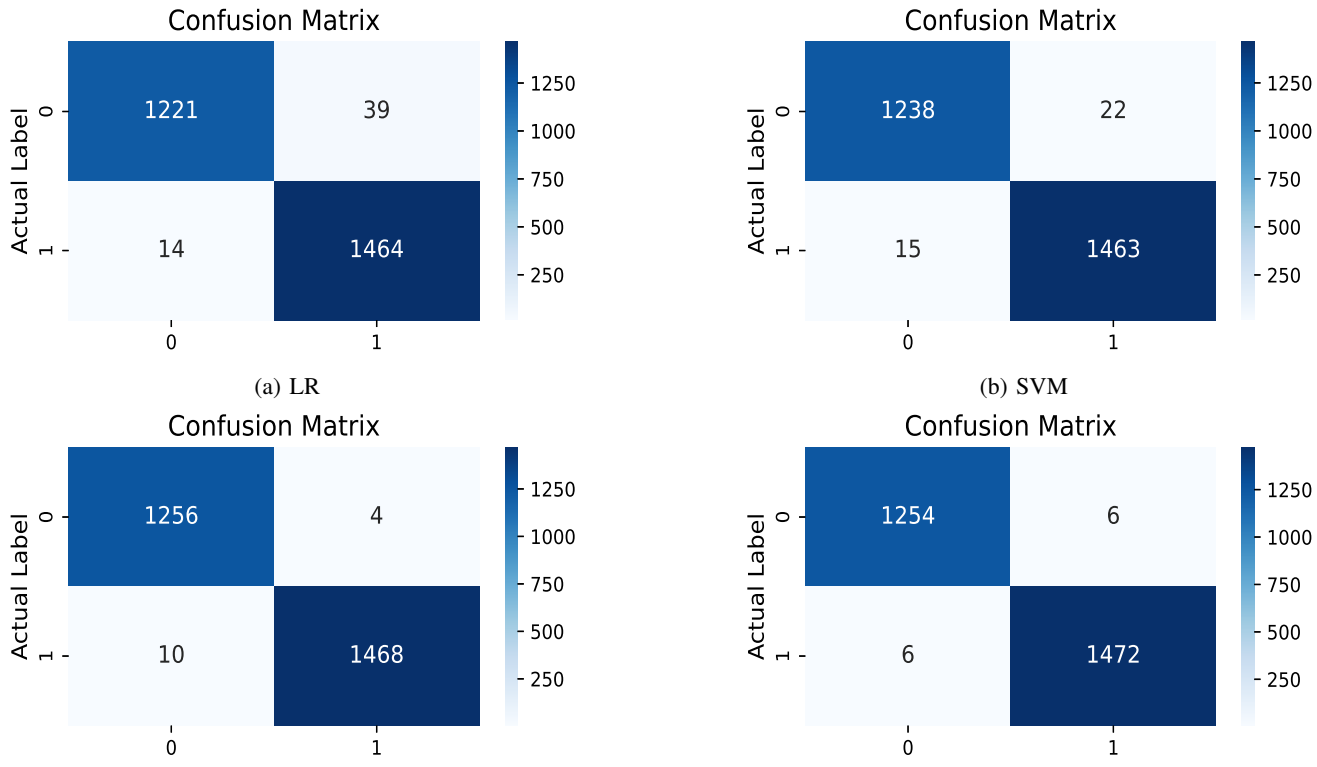
Furthermore, Receiver Operating Characteristic (ROC) curves and the corresponding Area Under the Curve (AUC) values were found to evaluate the proposed model's classification performance against the baseline model mentioned previously. The ability of the model to balance between the True Positive Rate (TPR) and the False Positive Rate (FPR) can be measured by these metrics, providing insights into its overall performance. It has been observed in Fig. 5 that the present approach accomplishes the highest result (AUC of 1.00) compared to the baseline models. This implies that the proposed classifier reaches the top-left corner (TPR is 1 and FPR is 0). Furthermore, this result has not been achieved by

TABLE IV: Model parameters

Parameter	Value
Batch size	128
Epochs	40
Image width	60
Learning rate	0.001

TABLE V: A comparison between the proposed method and existing methods

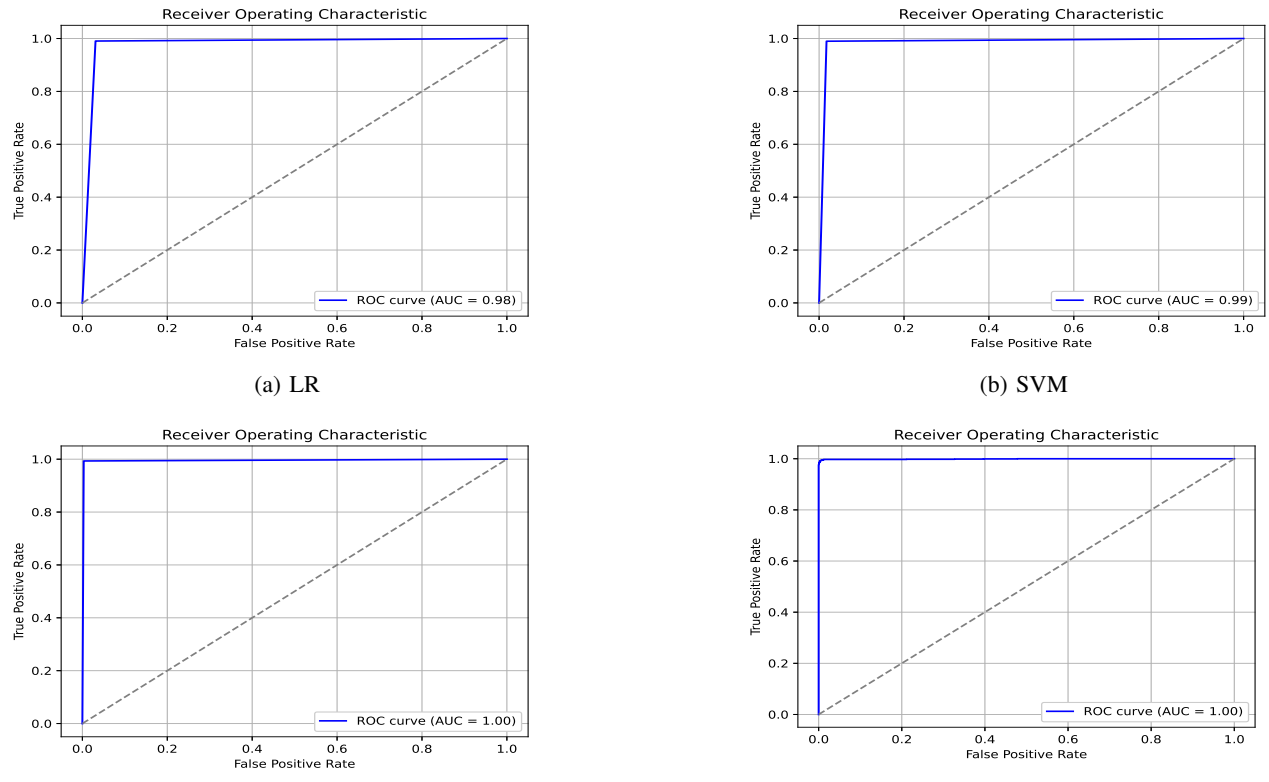
Method	Accuracy	Recall	Precision	F1-score
LR	0.9800	0.9800	0.9800	0.9800
SVM	0.9900	0.9800	0.9900	0.9900
RF	0.9900	0.9900	0.9900	0.9900
RF with USE-Word2Vec	0.9945	0.9926	0.9973	**0.9949**
CNN with USE-Word2Vec	0.9905	0.9946	0.9878	0.9912
Proposed method	**0.9956**	**0.9952**	**0.9966**	**0.9949**



(a) LR (b) SVM

(c) RF (d) The proposed approach

Fig. 4: Confusion Matrix for the testing dataset



(a) LR (b) SVM

(c) RF (d) The proposed approach

Fig. 5: ROC curves of our approach for XSS attack detection

any approach listed in [23].

A further experiment was conducted to assess training time, comparing the proposed method against existing approaches. As shown in Table VI, the proposed method requires less training time both per batch and per sample. A training time of 4 ms per sample suggests strong scalability,

TABLE VI: Training time for two different models

Method	Epoch level time (sec)	Sample level time (ms)
CNN with USE-Word2Vec	141.9	12.9
Proposed method	**45**	**4**

making it suitable for larger datasets and further model generalization. Additionally, online detection time is a critical factor in real-time web environments. Our method achieves an average detection time of 3 ms per sample, enabling rapid response to XSS attacks. This makes the approach highly effective for high-traffic web applications that demand both speed and precision in security.

VI. Conclusion

In this work, a method for detecting XSS vulnerabilities by reducing the detection problem to a binary classification task is proposed. A deep learning approach is adopted, specifically applying a CNN model for feature extraction, followed by a fully connected neural network for classification. Before classification, the script being analyzed is converted into a grayscale image and fed into the CNN. The performance of the proposed method was demonstrated using a dataset containing both benign and malicious scripts, introducing a learning model capable of detecting XSS vulnerabilities online. Experimental results show that this method achieves high accuracy and outperforms existing approaches. For future work, the plan is to extend this method to handle multiple types of XSS vulnerabilities. Specifically, given a script, the goal is to identify which specific XSS vulnerabilities it contains.

References

[1] Z. Elkhadir and M. A. Begdouri, "Enhancing IoT security: A comparative analysis of preprocessing techniques and classifier performance on IoT-23 and CICIoT-2023 datasets."

[2] S. M. Abdullah, "A two-phase analyzer for vulnerabilities of online social media users," *IAENG International Journal of Computer Science*, vol. 47, no. 2, pp. 144–153, 2020.

[3] F. Femi-Oyewole, V. Osamor, and D. Okunbor, "Ensemble machine learning approach for identifying real-time threats in security operations center." *IAENG International Journal of Computer Science*, vol. 51, no. 12, pp. 2094–2122, 2024.

[4] G. Wassermann and Z. Su, "Static detection of cross-site scripting vulnerabilities," in *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 171–180.

[5] M. Johns, B. Engelmann, and J. Posegga, "Xssds: Server-side detection of cross-site scripting attacks," in *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2008, pp. 335–344.

[6] H. Shahriar and M. Zulkernine, "S2XS2: a server side approach to automatically detect XSS attacks," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2011, pp. 7–14.

[7] ——, "Injecting comments to detect javascript code injection attacks," in *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*. IEEE, 2011, pp. 104–109.

[8] S. Gupta and B. B. Gupta, "XSS-SAFE: a server-side approach to detect and mitigate cross-site scripting (XSS) attacks in javascript code," *Arabian Journal for Science and Engineering*, vol. 41, pp. 897–920, 2016.

[9] R. Alhamyani and M. Alshammari, "Machine learning-driven detection of cross-site scripting attacks," *Information*, vol. 15, no. 7, p. 420, 2024.

[10] F. Younas, A. Raza, N. Thalji, L. Abualigah, R. A. Zitar, and H. Jia, "An efficient artificial intelligence approach for early detection of cross-site scripting attacks," *Decision Analytics Journal*, vol. 11, p. 100466, 2024.

[11] J. Kaur, U. Garg, and G. Bathla, "Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review," *Artificial Intelligence Review*, vol. 56, no. 11, pp. 12 725–12 769, 2023.

[12] G. Kaur, Y. Malik, H. Samuel, and F. Jaafar, "Detecting blind cross-site scripting attacks using machine learning," in *Proceedings of the 2018 International Conference on Signal Processing and Machine Learning*, 2018, pp. 22–25.

[13] S. Sharma, P. Zavarsky, and S. Butakov, "Machine learning based intrusion detection system for web-based attacks," in *2020 IEEE 6th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*. IEEE, 2020, pp. 227–230.

[14] S. Goswami, N. Hoque, D. K. Bhattacharyya, and J. Kalita, "An unsupervised method for detection of XSS attack." *Int. J. Netw. Secur.*, vol. 19, no. 5, pp. 761–775, 2017.

[15] Y. Fang, C. Huang, Y. Xu, and Y. Li, "Rlxss: Optimizing XSS detection model to defend against adversarial attacks based on reinforcement learning," *Future Internet*, vol. 11, no. 8, p. 177, 2019.

[16] I. Tariq, M. A. Sindhu, R. A. Abbasi, A. S. Khattak, O. Maqbool, and G. F. Siddiqui, "Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning," *Expert Systems with Applications*, vol. 168, p. 114386, 2021.

[17] C. Taoussi, I. Hafidi, and A. Metrane, "Machine learning and deep learning in health informatics: Advancements, applications, and challenges."

[18] F. Feng, X. Liu, B. Yong, R. Zhou, and Q. Zhou, "Anomaly detection in ad-hoc networks based on deep learning model: A plug and play device," *Ad Hoc Networks*, vol. 84, pp. 82–89, 2019.

[19] A. Ajiboye, M. Olumoye, D. Aleburu, A. Olayiwola, D. Olayiwola, and S. Ajose, "Dimensionality reduction for deep learning based intrusion detection systems for iot," in *Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists*, 2023, pp. 76–81.

[20] Y. Wu and X. Bao, "Anticipating lag synchronization based on BO-CNN-LSTM." *IAENG International Journal of Applied Mathematics*, vol. 55, no. 5, pp. 1325–1332, 2025.

[21] F. Mokbal, W. Dan, A. Imran, L. Jiuchuan, F. Akhtar, and W. Xiaoxi, "MLPXSS: an integrated XSS-based attack detection scheme in web applications using multilayer perceptron technique," *IEEE Access*, vol. 7, pp. 100 567–100 580, 2019.

[22] W. Melicher, C. Fung, L. Bauer, and L. Jia, "Towards a lightweight, hybrid approach for detecting DOM XSS vulnerabilities with machine learning," in *Proceedings of the Web Conference 2021*, 2021, pp. 2684–2695.

[23] R. Bakır and H. Bakır, "Swift detection of XSS attacks: Enhancing XSS attack detection by leveraging hybrid semantic embeddings and ai techniques," *Arabian Journal for Science and Engineering*, pp. 1–17, 2024.

[24] A. Hoffman, *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. O'Reilly Media, 2024. [Online]. Available: https://books.google.iq/books?id=aL7uEAAAQBAJ

[25] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in Neural Information Processing Systems*, vol. 2, 1989.

[26] L. Zhang, J. Lin, B. Liu, Z. Zhang, X. Yan, and M. Wei, "A review on deep learning applications in prognostics and health management," *IEEE Access*, vol. 7, pp. 162 415–162 438, 2019.

[27] S. Fadnavis, "Image interpolation techniques in digital image processing: an overview," *International Journal of Engineering Research and Applications*, vol. 4, no. 10, pp. 70–73, 2014.

Ahmed Al-Ajeli received his B.Sc. and M.Sc. degrees in Computer Science from the University of Babylon, Iraq, in 1999 and 2002, respectively. He worked as an assistant lecturer at the Department of Computer Science, the University of Babylon. In 2017, he received his Ph.D. in Computer Science from the University of Birmingham, UK. Currently, he holds an Assistant Professor position at the College of Information Technology, University of Babylon; and is the head of the Cyber Security Department. His current research interests include fault diagnosis/prognosis in discrete-event systems, machine learning and anomaly detection.