LAUTECH Journal of Engineering and Technology 17 (2) 2023: 111-120

PERFORMANCE EVALUATION OF HYBRID AND STANDALONE TECHNIQUES ON WEB APPLICATIONS BASED CROSS-SITE SCRIPTING ATTACKS

[1*]Ayogu B. A., [2]Ogunleye G. O., [3]Benjamin T. F. and [4]Ugwu J.N.

[1,2,,3,4]Department of Computer Science, Federal University Oye Ekiti, Nigeria
Corresponding Author, email: bosede.ayogu@fuoye.edu.ng

ABSTRACT

Cross-Site Scripting (XSS) is a type of malicious scripts that are broadcasted on the web applications through Hyper Text Transfer Protocol (HTTP). There are three categories of XSS: persistent, reflected, and Document Object Model DOM-based (Document Object Model). In the persistent attack, the malicious code is stored into the database and execute on every browser that loads the infected webpage. For the reflected attack, the user is tricked into submitting a form that sends malicious code to the victim's browser, while DOM-based attack is done by manipulating the user's DOM environment, which does not affect the HTTP response and the web server but the client side of the affected user. To shield users against data theft, this research targets to improve the detection accuracy of cross-site scripting attacks in web applications. The dataset underwent pre-processing. Pearson Correlation technique was used to choose sixty correlated features of sixty-eight features for the research. The efficacy of the CNN-LSTM hybridized approach and their standalone were demonstrated and evaluated using accuracy criteria. Experimental results recorded 99.87% for both hybrid and LSTM approaches respectively with margin of 0.1 lower than CNN, but higher in terms of other metrics, implying that all the approaches can be used for the detection of cross-site scripting attacks.

Keywords: *CNN, Feature selection, LSTM, Security vulnerabilities, XSS Dataset.*

INTRODUCTION

Web applications are popular software that can be used conveniently, and are accessible from any device with an internet connection. To access a web application over the internet, HTTP is the general protocol designed for this purpose. Web applications are developed in such a way that allows a large number of users to access it simultaneously without facing compatibility issues (Indeed Editorial, 2022). Categories of organization that use web application include, banking, education, and social media, which are trusted by millions of users. A web application with security vulnerability will always create risk for the application's users (Mack *et al.*, 2019). Cross-Site Scripting (XSS) is among the common attacks found to take advantage of vulnerabilities in web applications. This type of attack occurs when an attacker injects malicious code to compromise the interactions with users thereby taking advantage of the vulnerable application. The code can be executed to steal users' personal information such as cookies and sensitive data, and redirect users to dangerous websites (Gupta, & Gupta, 2017). The existence of XSS vulnerabilities can be traced back to when JavaScript was introduced for client-side application development which allows developers to create dynamic webpages that only execute on the client end (Garcia-Alfaro & Navarro-Arribas (2007); Aliga *et al.* (2018)). The development gave developers the ability to embed scripts inside webpages to perform a task such as validation of input form, adding interactive behaviours to webpages, and it reduces the load on the server (Guha *et al.*, *2010*). Attackers use the advantage of the scripting language to inject

malicious code into vulnerable web applications which are executed to steal users' sensitive data.

Cross-site scripting (XSS) attacks are of three types, namely: persistent, reflected, and document object model (DOM)-based. A persistent XSS attack occurs when malicious code is stored in a vulnerable web application's database and loaded onto every web browser that accesses the affected webpage. An example of this type of attack is when an attacker posts malicious code on their profile page on a social media application, and the code is saved in the database (Gupta & Gupta, 2017), whenever another user loads the infected webpage, the attacker is able to manipulate the victim's interaction with the webpage. The attacker can steal the victim's information like email passwords and other social media applications' login credentials.

In a reflected XSS attack, the victim is tricked into clicking a link or submitting a form and the injected malicious code is sent to the victim's web browser as a response to the server. In this type of attack, malicious code is not stored in the vulnerable web server but travelled through the web application uniform resource locator (URL) (Mack *et al*, 2019 and Gupta & Gupta, 2017) and executes on the victim's web browser. Attackers target web pages error messages or search results to inject their malicious codes.

DOM-based XSS attack is therefore when a malicious code is executed by manipulating the DOM environment in the victim's browser. In this type of attack, HTTP response is not changed neither is the web application server is affected. Modification is only done on the client side of the affected user, which can result to unexpected actions such as redirection to harmful websites (Parameshwaran *et al,* 2015).

The existence of vulnerabilities in web application can expose web users to attackers which steal sensitive information and cause unexpected damages. There are several available approaches to mitigate the effect of cyber-attacks in web applications, which have recorded high degree of positive outcomes (Isam et al, 2023 and Kumar & Ponsam, 2023). This research takes into consideration two deep learning methods (CNN and LSTM), implement them independently for the detection of cross-site scripting attacks in web applications to protect users from information thefts and also combine the standalone systems to have a hybrid system for checking the efficacy of the proposed deep learning methods using various standard metrics.

LITERATURE REVIEW

This section provides the existing researches on cross-site scripting attacks in web applications. Kadhim and Gaata (2020) combined CNN and LSTM to identify XSS in web applications. The XSS dataset was pre-processed and tokenized for Word2Vec. The experiments performed on this method yielded an accuracy rate of 99.4%. Alquari *et al* (2022) proposed a modular neural network to reduce the false-positive rate in the detection of XSS attacks in web application, to distinguish Benign words and Malicious words from the text payload, Word to Vector (Word2vec) and Continuous Bags of Words (CBOW) Models were employed. Fifty (50) features were used in this research. The experimental result showed that the proposed method achieved an accuracy of 99.96%. (Odun_Ayo, 2021) proposed a deep learning approach to build a secure system that can identify and stop cross-site scripting attacks in cloud-based web applications. A web application was developed with multi-layer perceptron deep learning model embedded on the server-side of the web application and hosted on the cloud. The experimental result of the system showed 99.47%

accuracy and 0.99% false-positive rate. Their future work is to further develop the model to counter and prevent other web-based attacks.

Mack *et al.* (2019) developed a simulation environment using XAMPP (XAMPP Apache + MariaDB + PHP + Perl) and VirtualBox to test four different types of XSS attacks: persistent, reflected, cookie-stealing, and keylogging. On a webpage hosted locally, each XSS attack was launched and analyzed both quantitatively and qualitatively. The evaluations were carried out using NIST CVSS 3.0 ratings, which identified whether the attack breaches integrity, availability, and confidentiality. To access the attacks, Base Score (BS) algorithm was used to calculate the scores for each attack. The BS formula's results revealed that a reflexed attack has the lowest effect, while keylogging has the highest effect. Their future work is to conduct similar studies on additional XSS variants. Garcia-Alfaro & Navarro-Arribas (2007) developed an intelligent tool for the detection of cross-site scripting defects in web applications. The implementation was carried out based on fuzzy logic. The system was tested on seven sources which XSS vulnerabilities can be introduced to web applications. The implemented system showed 95% accuracy and 0.99% false-positive rate. Their future work is to involve more DOM-based features that could lead to server-side injection vulnerabilities.

A system that can detect a malicious script and preventing it from storing it on the server was developed by Mereani & Howe (2018). The system was based on the mixture of classifiers using cascading to build a two-phase classifier and the stacking ensemble technique to improve detection accuracy. The initial phase of the classifier was to differentiate between plain text and script text. It retuned 100% accuracy, that is, no script text was classified as plain text. The second phase used Support Vector Machine as the meta level classifier and scored 99% accuracy. The limitation of the system is the inability to detect Base64 encoding. All the presented reviewed researches need for further research on XSS detection based on the provided limitations that affect the classification of the models. Thus, this is the motivating factor for this research.

METHODOLOGY

The methodology used in achieving the proposed systems is given in this section, which is divided into phases such as data description, feature selection, and experimental analysis of both standalone, and hybrid systems. Having a standalone system means that each of the deep learning approaches employed in this research are implemented independently without being bundled with one another, whist the hybrid approach requires bundling the two systems to achieve a common goal. The research employed CNN, and LSTM deep learn approaches to achieve the standalone, while the combination of both with sequential algorithm were used to form the hybrid method for detecting the XXS in web application.

System architecture

The architectural design of the proposed systems is shown in figure 1, comprising all the processes involved in the experimental analysis of the classification systems.

Data gathering and preprocessing

The study used a cross-site scripting dataset that was collated by Mokbal (2021). The dataset consists of one hundred and one thousand (101000) instances, split into training dataset and testing sets.

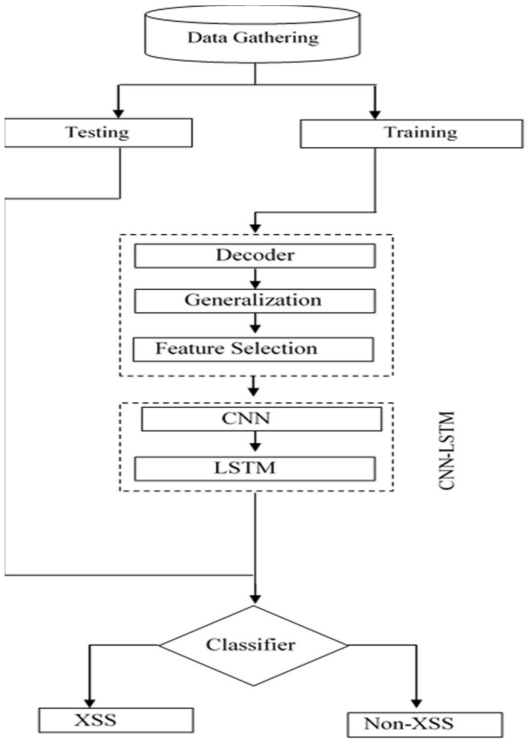


Figure 1. - Architecture of the proposed system

The training dataset consists of eight thousand and eight hundred thousand (80800) instances, which seven hundred and ninety-five (795) are attack instances, and eighty thousand and five (80005) are benign instances. The testing dataset consists of two thousand and two hundred (20200) instances, which two hundred and five (205) are attack instances, and nineteen thousand nine hundred and ninety-five (19995) are benign instances. The dataset sample is presented in Table 1.

Table 1: Samples of the datasets

	Training	**Testing**
Attack	795	205
Benign	80005	19995
Total	80800	20200

EXPERIMENTAL ANALYSIS

To make the data suitable for building and training deep learning models, there was a need for data cleaning and organizing which involves preprocessing such as decoding, generalization, and feature selection. To bypass traditional validation methods relying on regular expressions, attackers can encode their malicious codes using techniques such as URL encoding, Unicode encoding, Hex encoding, HTML entity encoding, among others. Therefore, this study employed a decoder capable of transforming encoded text into its normal textual representation. To handle data redundancy, decoded data is generalized by eliminating blank characters, "http://", https://, and www, this process sanitizes the data. To select the relevant features suitable for the classification systems, this research used a set of expressions for data categorization based on the scripting features. Pearson's Correlation method was used to evaluate the strength of relationship between features which ranges between -1 and +1. Thereafter, the selected features are input to the CNN and LSTM classification systems which were further combined to build a hybrid CNN-LSTM classification system for comparison purpose. The mathematical representations of the models employed in this research are given in equation 1 to 15.

Pearson's Correlation

$$p = \frac{n(\sum mj) - (\sum m)(\sum j)}{\sqrt{[n\sum m^2 - (\sum j)^2]\,[n\sum m^2 - (\sum j)^2]}} \quad (1)$$

where

$p = Pearson\ Coefficient$

$n = number\ of\ the\ pairs$

$\sum mj = sum\ of\ products\ of\ the\ pairs$

$\sum m = sum\ of\ the\ m\ scores$

$\sum j = sum\ of\ the\ j\ scores$

$\sum m^2 = sum\ of\ the\ squared\ m\ scores$

$$\sum j^2 = sum\ of\ the\ squared\ j\ scores$$

Sixty (60) out of seventy-seven (67) features were selected for the proposed system using Pearson's Correlation.

Convolutional neural networks (CNN)

Convolutional Neural Networks was motivated by the neurons in human brains. It is the most popular and used algorithm because of its ability to identifies important features without human interference (Mokbal, and Alzubaidi *et al*, 2021). Convolutional neural networks (CNNs) consist of multiple layers, including an input layer, convolutional layers, pooling layers, and an output layer. The input layer serves as the entry point for the image being processed. Convolutional layers extract the image features. Pooling layers, on the other hand, help to down-sample the feature map to reduce its dimensionality. Finally, the output layer performs the classification or recognition task.

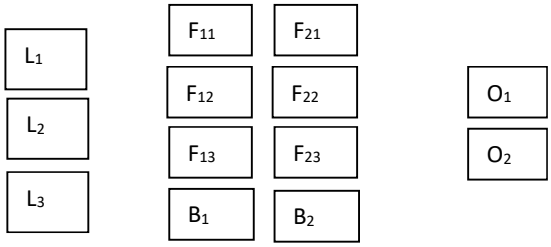
The mathematical notation for convolution is commonly denoted using an asterisk (*). When an input image is represented as L and a filter is represented as F, the equation for this operation would be written as:

$$O = L * F \quad (2)$$

$$Where\ O = Output\ \ L = Input\ Layer, and$$

$$F = Filter/Kernel$$

The filter iterates through different sections or patches of the image, multiplies the values of the elements of the filter and the image by each other element and then computes the sum of the resulting product. The calculation can be shown below:



The calculation can be shown below:

$$O_1 = B_1 + (L_1 * F_{11}) + (L_2 * F_{12}) + (L_3 * F_{13}) \quad (3)$$

$$O_2 = B_2 + (L_1 * F_{21}) + (L_2 * F_{22}) + (L_3 * F_{23}) \quad (4)$$

$$O_3 = B_1 + (L_1 * F_{31}) + (L_2 * F_{32}) + (L_3 * F_{33}) \quad (5)$$

$$O_d = B_d + (L_1 * F_{d1}) + (L_2 * F_{d2}) + (L_3 * F_{d3}) \quad (6)$$

The example above has the input with a shape of (3, 3) and the filter has a shape of (2, 2). Given these dimensions for the image and filter, the output matrix will have a shape of (2, 2). Concisely,

$$O_i = B_i + \sum_{j=1}^{n} L_j * F_{ij}, i = 1 \ldots d \quad (7)$$

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \\ \vdots \\ O_d \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_d \end{bmatrix} + \begin{bmatrix} F_{11} & F_{12} & F_{13} & \cdots & F_{1n} \\ F_{21} & F_{22} & F_{23} & \cdots & F_{2n} \\ F_{31} & F_{32} & F_{33} & \cdots & F_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{d1} & F_{d2} & F_{d3} & \cdots & Fdn \end{bmatrix} * \begin{bmatrix} L_1 \\ L_2 \\ L_3 \\ \vdots \\ L_n \end{bmatrix} \quad (8)$$

In a simple equation,

$$O = B + F * L \quad (9)$$

CNN-LSTM

LSTM is a variation of RNN and was first introduced by Hochreiter and Schmidhuber, with further improvements made by other researchers. These networks are the most used type of RNN because they are found to be effective for a several range of problems (Aliga *et al,* 2018). LSTM networks are advanced method for sequence learning. While they are not frequently used for time series forecasting, they are well-suited for this field due to their ability to learn long-term dependencies. The CNN based

LSTM was used to increase the accuracy of the network by using the Convolutional Neural Network (CNN) one-dimensional three-time and zero padding two-time, resulting from concatenation (CNN1, CNN2 and CNN3) use in the LSTM. The mathematical representation of the theory on how the hidden state hs-s time s is determined from the previous hidden state hs-1 which is behind the LSTM units' concepts is shown below:

Input gate is displayed as

$$l(s) = \delta\big(W(l)x(s) + u(l)h(s-1)\big) \quad (10)$$

Forget Gate computed as,

$$g(s) = \delta(W(g)x(s) + u(g)h(s-1) \quad (11)$$

and, Output gate computed as,

$$o(s) = \delta(W(o)x(s) + u(o)h(s-1) \quad (12)$$

also, the new memory cell as,

$$c^{\sim}(s) = \tanh\big(W(c)x(s) + u(c)h(s-1)\big) \quad (13)$$

overall, the final memory cell is calculated as,

$$c(s) = g(s)oc^{\sim}(s-1) + l(s)o\ c^{\sim}(s) \quad (14)$$

and,

$$h(s) = o(s)o\ \tanh(c(s)) \quad (15)$$

$where\ s = time, and\ l = input$

The structure of CNN-LSTM network is shown in Figure 2:

RESULTS AND DISCUSSION

The XSS dataset consist of 101000 instances made up of 100000 normal and 1000 attacks instances with 67 features. The training dataset contains 70% of the dataset collated and the testing dataset contains 30% of the dataset collated. Pearson's Correlation method was applied on both training and testing sets to reduce the input features to 60, whilst the testing dataset was further used to determine the efficacy of the proposed models.

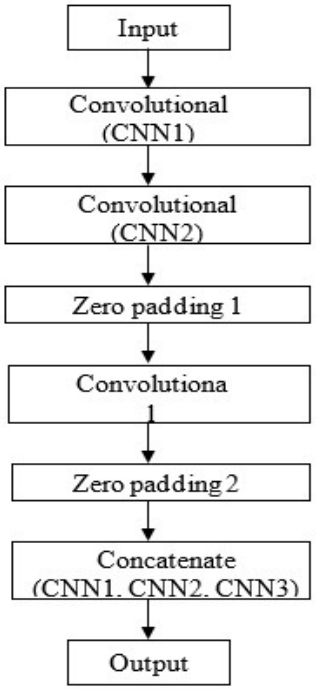


Figure 2: Structure of CNN-LSTM

These processes were performed on the hybridized model (CNN-LSTM) and each method individually. The implementation was done using Python 3.6 programming language. The confusion matrix for both standalone and hybridized methods from python library is shown in Figure 3, while Table 2 shows the overall performance of the models and its corresponding graphs based on the standard metrics are given in Figure 4 and 5, respectively. The parameters used to evaluate the metrics are;

$$Accuracy = \frac{True\ Positive\ (TP) + True\ Negative\ (TN)}{TP + TN + False\ Positive\ (FP) + Fal\quad Negative\ (FN)} \quad (16)$$

$$Precison = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + Fals\quad Positive\ (FP)} \quad (17)$$

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + Fals\quad Negative\ (FN)} \quad (18)$$

$$F1\ Score = 2 * \frac{Precison * Recall}{Precison + Reca} \quad (19)$$

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(testY, y_pred.round())

    print(cm)

    [[19989     6]
     [   17   188]]
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(testY, y_pred.round())

    print(cm)

    [[19978    17]
     [    7   198]]
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(testY, y_pred.round())

    print(cm)

    [[19989     6]
     [   17   188]]
```

Figure 3: Confusion matric for both standalone and hybridize methods

Table 2: Overall performance of all the models

Model	True Positive (TP)	False Positive (FP)	True Negative (TN)	False Negative (FN)	Accuracy	Precision	Recall	F1-Score
CNN-LSTM (Hybridized)	19989	17	188	6	99.87%	99.92%	99.97%	99.95%
CNN	19978	17	198	7	99.88%	99.91%	99.97%	99.94%
LSTM	19989	17	188	6	99.87%	98.92%	99.97%	99.95%

The confusion matrix comprises four classes, including True Positives (TP) representing correctly classified XSS samples, False Positives (FP) representing non-XSS samples wrongly classified as XSS, True Negatives (TN) indicating appropriately classified non-XSS samples, and False Negatives (FN) representing XSS samples inappropriately classified as non-XSS. For the hybridized model, out

of the 19995 of Benign records, 19989 were correctly classified, while 17 were misclassified as attacks, while the total number of attacks misclassified as Benign were 6 out of the 205 attacks present in the testing set. The CNN misclassified 17 and 7 as attacks and benign instances, respectively, whilst LSTM model recorded the same result as hybridized approach achieving 99.87% accuracy which means CNN outperformed both methods in terms of accuracy achieving 99.88% accuracy. The graphical representation of the proposed models based on the standard metrics employed in this research are shown in Figure 4 and 5, respectively.

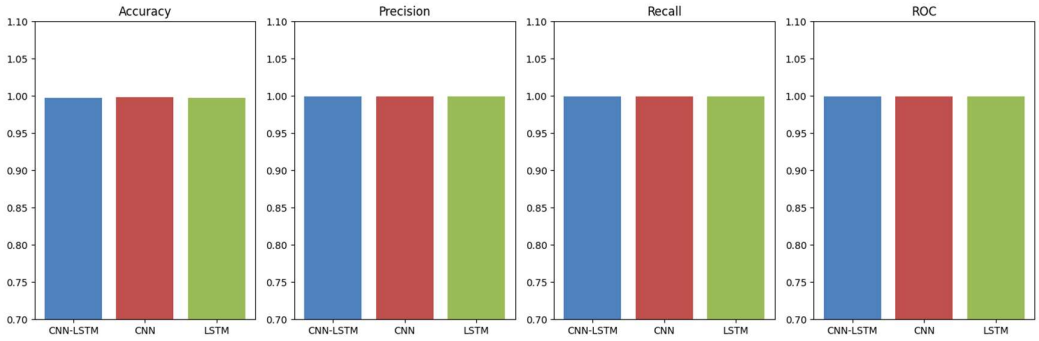


Figure 4: Overall performance of all the models

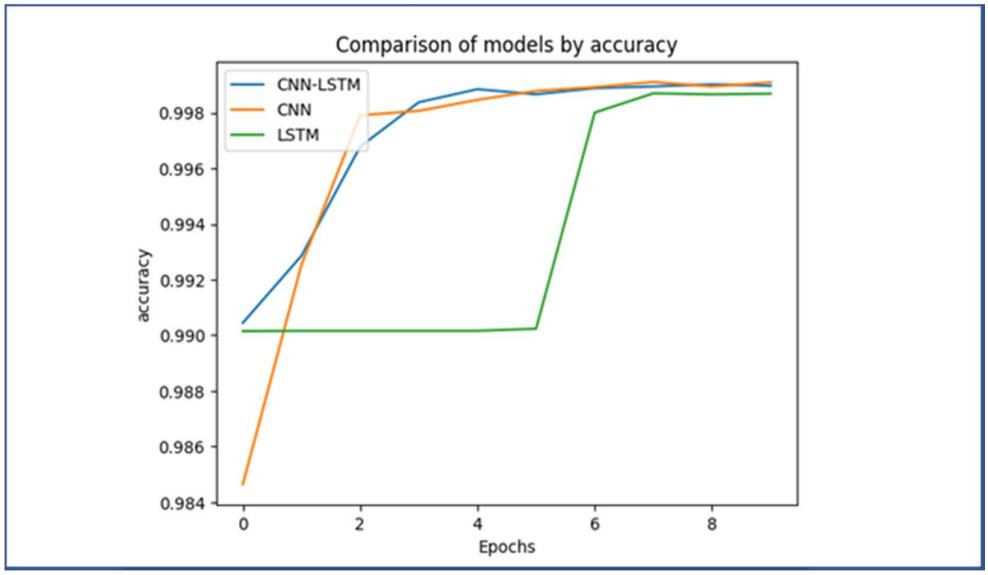


Figure 5: Comparison of models by accuracy

CONCLUSION

To shield users against data theft, this study offers an effective technique to detect cross-site scripting attacks in web applications. The proposed method was carried out using the Python programming language. The experiments were demonstrated on the hybridized, and standalone approaches. Experimental evaluation performance showed that either the hybridized or standalone approaches can be used for the detection of cross-site scripting attacks effectively. All the models achieved more

than 99% performance in terms of the evaluation matrics employed in this research.

REFERENCES

Aliga, P. A., John-Otumu, A. M., Imhanlahimi, R. E., and Akpe, A. C. E (2018). Cross Site Scripting Attacks in Web-Based Applications. Journal of Advances in Science and Engineering, 1(2), 25-35.

Alqarni, A., Alsharif, N., Khan, N., Georgieva, L., Pardede, Eric., Alzahrani, M., and Ahmad, N (2022). MNN-XSS: Modular neural network-based approach for XSS attack detection. Computers, Materials and Continua, 70(2), 4075-4085.

Alzubaidi, L., Duan, Y., Al-Dujaili, A., Ibraheem, I. K., Alkenani, A. H., Santamaria, J., Fadhel M. A., Al-Shamma, O., and Zhang, J. (2021). Deepening into the suitability of using pre-trained models of ImageNet against a lightweight convolutional neural network in medical imaging: An experimental study. Peer J Computer Science, 7, e715.

Ayeni, B. K., Sahalu, J. B. and Adeyanju, K. R. (2018). Detecting cross-site scripting in web applications using fuzzy inference system. Journal of Computer Networks and Communications.

Garcia-Alfaro, J. and Navarro-Arribas, G. (2007). Prevention of cross-site scripting attacks on current web applications. In On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS: OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part II (pp. 1770-1784). Springer Berlin Heidelberg.

Guha, A., Saftoiu, C. and Krishnamurthi, S. (2010). The essence of JavaScript. In ECOOP 2010–Object-Oriented Programming: 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings 24 (pp. 126-150). Springer Berlin Heidelberg.

Gupta, S. and Gupta, B. B. (2017). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. International Journal of System Assurance Engineering and Management, 8, 512-530.

Indeed Editorial Team, (2022) What Is a Web Application? How It Works, Benefits and Examples, Retrieved December 16, 2022, from https://in.indeed.com/career-advice/career-development/what-is-a-web-application

Kadhim, R. and Gaata, M. (2020). A hybrid of CNN and LSTM methods for securing web application against cross-site scripting attack. Indones. J. Electr. Eng. Comput. Sci, 21, 1022-1029.

Kumar, J. H., and Ponsam, J. J. G. (2023). Cross Site Scripting (XSS) vulnerability detection using Machine Learning and Statistical Analysis. International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India. (pp. 1-9).

Mack, J., Hu, Y. H. F. and Hoppa, M.A. (2019). A Study of Existing Cross-Site Scripting Detection and Prevention Techniques Using XAMPP and VirtualBox. Virginia Journal of Science, 70(3), pp. 1-23.

Mereani, F. A. and Howe, J. M. (2018). Detecting cross-site attacks using machine learning. In The International Conference on Advanced Machine Learning Technologies

and Applications (AMLTA2018) (pp.200-210). Springer International Publishing.

Mokbal, F. M. M (2021). "Cross-site scripting attack(XSS) dataset," Figshare, 2021. [Online].Available:https//figshare.com/articles/dataset/XSS_dataset1_csv/13046138?file=24959207.

Odun-Ayo, I., Toro-Abasi, W., Adebiyi, M., and Alagbe, O. (2021). An implementation of real-time detection of cross-site scripting attacks on cloud-based web applications using deep learning. Bulletin of Electrical Engineering and Informatics, 10(5), 2442-2453.

Parameshwaran, I., Budianto, E., Shinde, S., Dang, H., Sadhu, A., and Saxena, P. (2015). DexterJS: robust testing platform for DOM-based XSS vulnerabilities. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 946-949).

Thajeel, I. K., Samsudin, K., Hashim, and S. J., Hashim, F. (2023). Machine and Deep Learning-based XSS Detection Approaches: A Systematic Literature Review. Journal of King Saud University - Computer and Information Sciences, 35(7), 1319-1578