Automatic Detection of HTTP Injection Attacks using Convolutional Neural Network and Deep Neural Network

Victor Odumuyiwa* and Analogbei Chibueze

Department of Computer Science, University of Lagos, Nigeria
E-mail: vodumuyiwa@unilag.edu.ng; analogbeichibueze@gmail.com
**Corresponding Author*

Received 05 April 2020; Accepted 31 August 2020;
Publication 06 February 2021

Abstract

HTTP injection attacks are well known cyber security threats with fatal consequences. These attacks initiated by malicious entities (either human or computer) send dangerous or unsafe malicious contents into the parameters of HTTP requests. Combatting injection attacks demands for the development of Web Intrusion Detection Systems (WIDS). Common WIDS follow a rule-based approach or a signature-based approach which have the common problem of high false-positive rate (wrongly classifying malicious HTTP requests) hence making them restricted to only one type of web application. They are easily bypassed and unable to detect new kinds of malicious attacks as they lack a sufficient model of understanding the representations of HTTP request parameters. In this paper, deep learning techniques are used to develop models that would automatically detect injection attacks in HTTP requests. A special layer called the character embedding layer in the deep learning models is used to allow the learning of the representation of the request parameter of HTTP requests in higher abstract levels and also aid in learning the relationships between the characters of the request parameter. The experimentation results showed that with deep learning, better injection attack detection is possible and given the right dataset, a deep learning

Journal of Cyber Security and Mobility, Vol. 9_4, 489–514.
doi: 10.13052/jcsm2245-1439.941

detection model would be able to correctly classify HTTP requests for any web application.

Keywords: HTTP, injection attack, DNN, CNN, cyber security, deep learning.

1 Introduction

The Internet has come a long way from a research project called ARPANET in 1963 to a widely accessible inter-network by billions of users around the world. Many inventions today revolve around the use of the Internet. Due to the rich nature and fast-growing availability of the Internet, the Internet has now become a place for people to store valuable information and to present that information in an organized format using web technologies. This allows many users around the world to have access to information and also guarantee the integrity of that information, but like all good things, the Internet has its own share of security flaws.

Web Services run on the Internet and are mostly built on the client-server architecture, where a dedicated computer acts as the server to various clients, providing the necessary information requested by the clients. Attacking web services has become prevalent and also turning to a lucrative venture by malicious individuals, either human or computer (bot), who try to obtain control of the server by sending malicious requests to the computer server. The increase in web attacks is directly or indirectly related to the increase of users of web applications and such increase will continue as observed by some cyber security blogs predicting that by 2022, 75% of the 8 billion projected world population would be Internet users.

This study focuses on one of the numerous types of web attacks on the Internet, which is HTTP injection attacks. HTTP (Hypertext Transfer Protocol) is an application layer protocol that does not keep state [1]. It communicates via a TCP channel and is about a series of requests for a resource and responses from a server. Hackers exploit the HTTP requests by injecting harmful code or resources in order to gain access to the server or have unauthorized access to information stored on the server. This exploitation is called HTTP injection attack and is categorized as malicious HTTP requests. There are many kinds and forms of injection attacks; the most common, being SQL (Structured Query Language) injection, involves the execution of malicious SQL database commands due to poorly validated data flowing from the client to the database server [2, 3].

In combatting injection attacks, solutions like intrusion detection systems for web applications were developed. Intrusion detection systems are software or devices that monitor networks or systems to find malicious activity or violations, which are collected or reported to an administrator. Intrusion detection systems identify suspicious-looking activities (in this case "HTTP requests"), block such requests from reaching the webserver and reports to the administrator. The goal of HTTP injection attack detection is to guarantee data integrity, confidentiality and availability on the web server so that users of the service can be rest assured that their information is safe.

Automatic detection of HTTP injection attack is a form of injection attack detection such that the attack is identified as soon as it is sent to the webserver. Unlike other detection approaches that identify the attack after the request has been executed, it is normally implemented via an intrusion detection system on either the server or client-side. It is an extra layer of security after data validation has been carried out on a web request.

Existing approaches in detecting HTTP injection attacks employ statistical methods and machine learning techniques for malicious HTTP requests detection. This study takes it further by making use of deep learning techniques [4, 5] to achieve better HTTP injection attack detection.

2 Related Works

Injection attacks are very dangerous to organizations and Web users. Combatting injection attacks demands for the development of Web Intrusion Detection Systems (WIDS). There are 3 popular approaches to building a WIDS. They are signature-based approach, rule-based approach and anomaly-based approach. Signature-based detection requires a library that holds malicious symbols in which the contents of the requests are checked to see if any symbol in the library appears in the request. If so, the request is deemed malicious. This method is very static, as it does not cater to new types of attacks and can fail to detect attacks which do not contain symbols in their library. Rule-based detection is like signature-based detection, but it does not keep a library of malicious symbols. Instead, it checks the HTTP request on some already defined rules that a benign HTTP request follows and if the request breaks one of the rules or a number of rules, it is considered malicious. Anomaly-based detection involves the training of a mathematical model to characterize the web requests so as to filter out malicious web requests. Unlike signature-based detection, this method can detect new types of attacks.

Several attempts have been made in detecting HTTP injection attacks and other forms of attacks. Gallagher and Eliassi-Rad [6], using anomaly-based detection approach, developed an HTTP attack classifier based on the vector space model, using TF-IDF (Term Frequency-Inverse Document Frequency) weights and cosine similarities to identify requests that are "valid" or "attack." The classifiers also went further to identify the type of attack (i.e. SQL injection, XSS, etc.). The model was evaluated using the ECML/PKDD 2007 dataset. The proposed model first trains by dividing the training requests into the various types of attacks and one set of valid request, then computes the TF-IDF weights for the requests after tokenizing the requests on special characters and encoded URL characters. The weights are then saved and used to classify test requests by computing the cosine similarity between their TF-IDF weights. They reported that the model achieved high precision and recall than previous methods in the literature.

Choudhary and Dhore [7] considered code injection attacks to be very fatal to Internet users and proposed a signature-based model to classify HTTP requests as either query-based or scripted and also detects the type of attack on the request if any. The proposed model consists of two main modules called query detector and script detector, which analyze the request independently. The request first goes through the query detector where it is analyzed if it has an invalid character before sending it to the script detector. If any invalid character is found, it is classified as malicious and does not go any further in the system. Once a request is considered valid, it goes to the script detector, which encodes any invalid tags found in the request before allowing the server to take action on the request. The proposed model was able to beat alternative approaches by detecting more forms of SQL injection attacks and all forms of XSS attacks.

Lampesberger et al. [8] also using an anomaly-based detection approach, presented a generic method for detecting anomalous and potentially malicious web requests without prior knowledge or the need for training data of the web-based application. It made use of a Markov model which was generated from legitimate sequences of semantic entries of web requests, such that a request that deviates from the sequence will be reported as anomalous, and was found to have acceptable false alarm and detection rates.

Kozik et al. [9] made use of a combination of anomaly-based detection and signature-based detection. Their proposed method involved a whitelist of legitimate requests. The whitelist is implemented as a hash-map with the key-value being a concatenation of the HTTP request method and the HTTP request URL. If a request's generated key is in the whitelist, the parameters of

the requests are encoded into a feature vector and a classifier is applied on the feature vector to determine if it is malicious or not and if the generated key of the request is not in the whitelist, the request is considered to be malicious. Their method was evaluated using the CSIC 2010 HTTP dataset.

Seyyar et al. [10] approached the issue of HTTP attacks by first identifying vulnerability scan checks by malicious users. These scans, on their own most often, are not attacks but they are used to locate vulnerability in the web application. Using access log files of web servers, they made use of signature and rule-based methods to efficiently detect web attack scans and additionally XSS and SQL injection attacks. Their method was reported to have a high detection probability and a low false alarm probability. They concluded that static rules are able to detect web vulnerability scans and that their method performs better on larger data sets.

Dong and Zhang [11] proposed an adaptive learning system called AMODS for detecting malicious queries. According to the authors, it follows the anomaly-based detection approach but allows for re-learning or model update which in turn reduces the False Positive Rate (FPR). AMODS included what was called SVM HYBRID which reduces manual work by choosing important queries and incorporating them into the training pool to update the detection model for labeling via adaptive learning. It was shown that AMODS outperforms SVMAL (Support Vector Machine Adaptive Learning) in both detection performance and the number of malicious queries obtained.

Althubiti et al. [12] compared different machine learning techniques in detecting web intrusion. The various techniques were applied to the CSIC 2010 HTTP dataset for intrusion detection purposes. Compared to other researches on intrusion detection, using the CSIC 2010 dataset, they went further to select the best five features using Weka. This brought about better results, high accuracy and reduced training time, which leads to the opportunity of semi-supervised learning approaches on the same CSIC 2010 dataset and also the usage of those five features on other datasets.

Rong et al. [13] proposed a malicious request detection system using the anomaly-based detection approach with re-learning ability like that of [11] but based on an improved Convolutional Neural Network (CNN) model. The CNN model improvement involved the addition of a character-level embedding layer and the addition of modified filters able to extract fine-grained features of the web request query string. They also compared their model with other traditional models like SVM and random forests and discovered that their model pays more attention to the extraction of the local feature in the

query string. The results showed that their improved CNN model outperforms traditional models on test datasets, although it has not been tested on practical applications at the time of the report.

3 Model

The goal of this study is to create a deep learning model capable of performing automatic detection of injection attacks in HTTP requests.

3.1 Conceptual Design

Three datasets were used in this study. The first two datasets are the ECML/PKDD 2007 dataset and the CSIC 2010 dataset. The third dataset was generated by combining the malicious payloads gotten from the PayloadsAllTheThings [14] project with both the ECML/PKDD 2007 and CISC 2010 datasets to form a new dataset which we named Hybrid 2020 dataset. More details about the datasets are provided in Section 4.3. The data gotten from each dataset was preprocessed in order to extract the relevant features needed for the study, the extracted features included the query parameter for GET requests and the body parameter for POST requests. After extracting the necessary features, they are further processed by converting a request parameter to a sequence of indexes from a predefined dictionary where an index corresponds to a particular character, the generated index sequence is passed as input to the detection model. This study made use of two deep learning models, the first being a deep neural network with an embedding layer and the second a convolutional neural network also with an embedding layer.

3.2 Detection Model

The goal of this study is to efficiently classify HTTP requests as injection attacks (malicious) or safe requests (benign), using a modified form of word embedding called character embedding in which an embedding table is built for every character instead of word. A deep neural network and a convolutional neural network are then used to extract the features and perform classification, and each model returns a detection result.

3.2.1 Character Representation and Embedding

A web request parameter is a sequence of characters. These characters fall under 3 main classes of characters: numeric characters, alphabetic characters,

and special characters. In total, there are 95 unique characters that can be found in an HTTP request parameter. In word embedding, each word is replaced with its index in the dictionary of possible words. Similarly, in character embedding, each character C_i is replaced with its index in the dictionary of possible characters, in this case from 1–95. This representation is then sent to the embedding layer.

3.2.2 Embedding Layer

The embedding layer takes in the index representation of the characters and returns a matrix of real numbers. For every character index in the character index sequence, there is a corresponding k-dimensional vector e_i from an embedding table E that embeds it. The embedding table is a matrix of N rows, where N is the number of unique characters possible for an HTTP request (i.e. 95) plus an entry for unknown characters. Each row in the embedding table is a vector representation of a character given its index from the dictionary of characters. The embedding table is first initialized with random values and the values are optimized via back-propagation which means that the model would be able to learn the relationships between every character.

The embedding layer is a 2-dimensional matrix with M rows where M is the number of characters to be considered for each request parameter and N columns where N is the same as the number of rows in the embedding table. If the request being considered has more than M characters the first M characters are used and if the number of characters in the request is less than M characters, the request parameter is padded with some "padding" characters to make the number of characters equal to M.

3.2.3 Deep Neural Network

Our DNN architecture is shown in Figure 1. The Input to the model is the index representation of the HTTP query gotten from the character dictionary, the embedding table then provides an embedding vector for each index making up the index representation and the matrix formed becomes the embedding layer. The embedding layer is flattened and the flatten layer which is a one-dimensional representation of the embedding matrix is formed, it then serves as the input layer to the feed forward neural network which is connected to a series of dense fully connected hidden layers. An output layer is added just after the last hidden layer which gives the prediction results of the computation, back propagation is used for optimization, the optimization allows for the embedding table vectors to be updated to better represent each character in a HTTP request. The RELU activation function is used in the

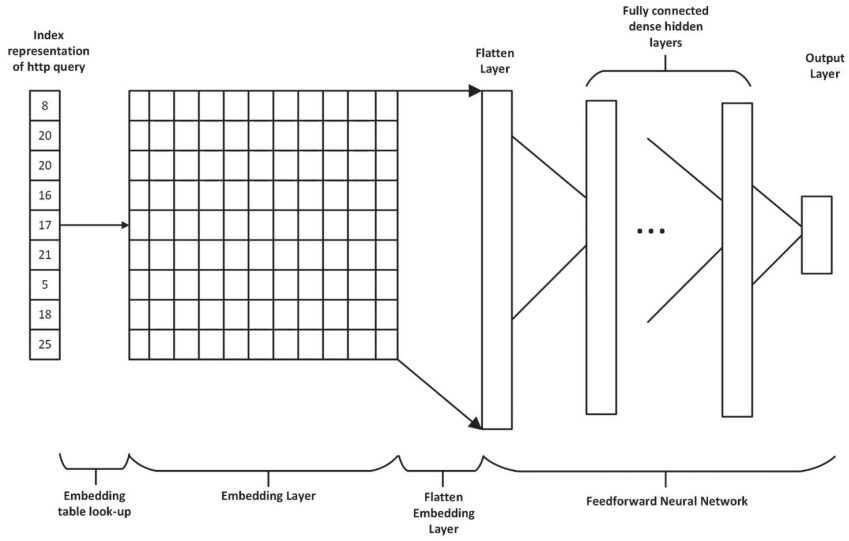

Figure 1 Deep neural network detection model architecture showing the embedding layer and feed forward neural network.

neuron layers in the hidden layers and the softmax activation function is used in the final dense neuron layer and leads to the output layer.

The DNN model used consists of 2 hidden layers (first with 3000 neurons and the second with 1024 neurons), Dropout was applied to the two hidden layers with the probability of retention $p = 0.5$. The reason this configuration was chosen was based on the need to avoid overfitting while still achieving good performance. Given the current hardware limitations, this configuration led to about 245 million trainable parameters which was the best possible.

3.2.4 Convolutional Neural Network

Our CNN architecture is shown in Figure 2. The input layer is a vector consisting of the index representation of the http request, the embedding layer is formed by the creation of the embedding matrix via lookup from the embedding table. The convolutional layer network, is a network of 6 convolutional layers all with 256 features (filters), the first 2 convolutional layers have a filter size of 7 and the other convolutional layers make use of a filter size of 3, the pooling operation is carried out in the first two layers and the last layer, each operation makes use of a pooling size of 3. Following from the pooling operation carried out on the last convolutional layer in the convolutional layer network, the resulting matrix is flattened and the resulting

one-dimensional matrix forms the flatten layer. The flatten layer is made as input to the fully connected layer network which consists of 2 dense layers with 1024 neurons and dropout of 0.5 leading to an output layer with 2 neurons which gives the results of the entire computation. Back propagation is used for optimization, the optimization allows for the embedding table vectors to be updated to better represent each character in a HTTP request.

The filter set chosen for the convolutional neural network configuration was motivated by the need to reduce the total number of training parameters going to the fully connected layers without affecting the accuracy. The 2 fully connected layers were chosen to avoid overfitting because the convolutions have reduced the dimensions of the input data, thereby exposing the more relevant features. An increase in the number of connected layers or convolutional layers may prove fatal as the model would only tightly fit the training set and would perform poorly in generalization to other examples. Other setups were considered but were not chosen due to the fact that the number of trainable parameters for those model configurations were higher than that of the chosen model configuration (9 million) and also because of the need to avoid overfitting.

An example of a different network configuration tried was to feed the output of the embedding layer to 3 different copies of the convolutional layer network configuration (as described above) and combining the flattened output of each configuration to a single layer which serves as input to a fully connected layer network with 2 dense layers each with 1024 neurons and dropout of 0.5. Although this configuration captures more features than the chosen configuration, however it has the limitations of high number of trainable parameters (51 million), overfitting and duplication of features where by a feature is considered twice which is not optimal.

4 Experimentation and Results

4.1 Test Environment

The models' architectures were implemented in python 3 using Keras [15, 16] with Tensorflow [17] backend and other third party libraries like sci-kit learn [18], pandas [19] and numpy [20].

4.2 Design of Experiments

In this study, three datasets were considered, the ECML/PKDD 2007 [21], CSIC 2010 [22], and the Hybrid 2020 datasets. The three datasets covered

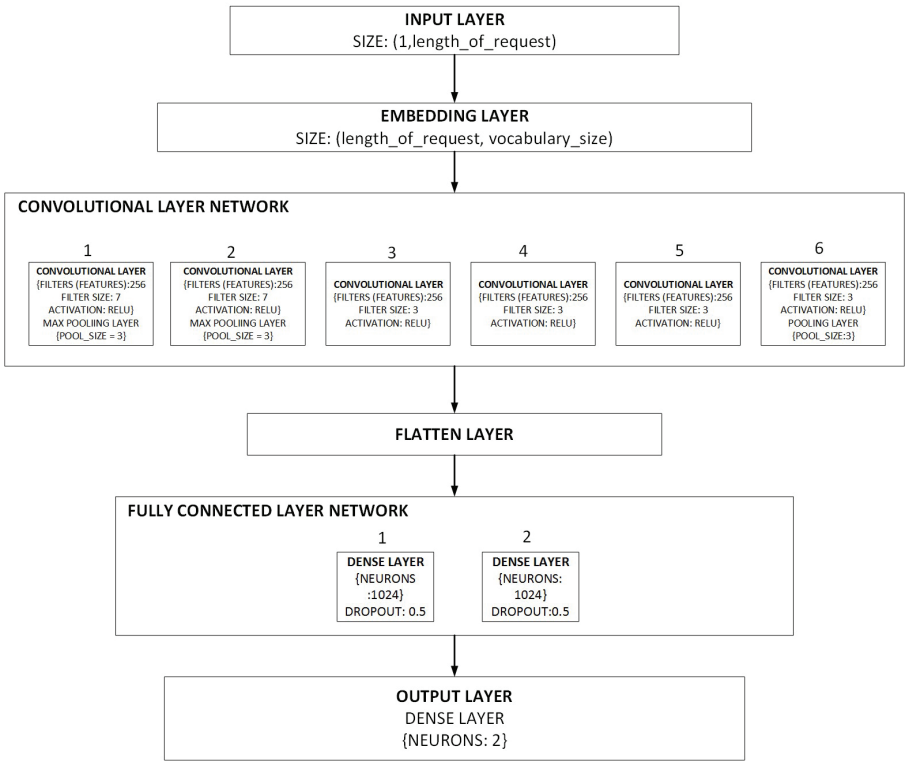

Figure 2 Block diagram of the convolutional neural network detection model architecture.

a larger range of HTTP injection attacks such as SQL injection, command injection and cross-site scripting. The composition of the three datasets is shown in Tables 1–4.

The Hybrid 2020 dataset was generated in the course of this study by combining the other two datasets with the malicious payloads gotten from the PayloadsAllTheThings [14] project hosted on GitHub which is an open source repository for malicious payloads of different types of HTTP, web and cloud based attacks that can be used for penetration testing to measure the security of web applications. The PayloadsAllTheThings repository is an updated repository of all forms of attack payloads and that is the reason for naming the generated dataset from it as Hybrid 2020. The dataset was created by converting known valid requests from both the ECML/PKDD 2007 and the CSIC 2010 datasets to malicious type by randomly replacing a value in

Table 1 Composition of the CSIC 2010 dataset used for the experiment

	Count
Malicious	19586
Benign	16000
Total	35586

Table 2 Composition of the ECML/PKDD dataset used for the experiment

		Count	
Valid		28962	
SSI		1818	
SQL injection		2221	
Xpath injection		2228	
path traversal		1983	
command injection		1985	
LDAP injection		2232	
XSS		1686	
Total	Benign	Malicious	Total
	28962	14153	43115

Table 3 Composition of the generated dataset

		Count	
SQL injection		5326	
command injection		1960	
LDAP injection		105	
XSS		6900	
Total	Benign	Malicious	Total
	0	14291	14291

the sequence of key-value pairs of the request with a malicious payload; this process resulted in 14,291 malicious request examples as shown in Table 3. This 14,291 generated malicious requests were added to the original ECML/PKDD dataset to form a new dataset containing a total of 57406 requests made up of 28962 benign requests and 28444 malicious requests as shown in Table 4. This also implies that the generated Hybrid 2020 dataset is a balanced dataset as compared to the original ECML/PKDD 2007 dataset.

Table 4 Composition of the hybrid 2020 dataset used for the experiment

		Count	
Valid		28962	
SSI		1818	
SQL injection		7547	
Xpath injection		2228	
path traversal		1983	
command injection		3945	
LDAP injection		2337	
XSS		8586	
Total	Benign	Malicious	Total
	28962	28444	57406

4.3 Experimentation Results

4.3.1 Performance Results with the CSIC 2010 Dataset

The dataset was divided with 70% (24910) used for training and 30% (10676) used for validation. The CNN model achieved a training accuracy of 86.04% and validation accuracy of 96.27%, the DNN model achieved a training accuracy of 85.52% and validation accuracy of 86.26%. The results from the various experiments carried out on this dataset are shown in Tables 5–8.

Table 5 Performance results from training and testing the CNN model on the CSIC 2010 dataset

Experiment	Precision (%)[1]	Recall[2]	F-measure	FPR[3]	Accuracy (%)
On 10676 requests (malicious and benign)	98.83	0.95	0.97	0.020	96.39

[1]The performance measures used in evaluating the performance of each model on the different datasets with both malicious and benign requests are precision, F-measure, recall, accuracy and false positive rate (FPR).
[2]Recall, also known as the True positive rate (TPR), shows the ability of the model to correctly classify a malicious request as malicious.
[3]False positive rate (FPR) shows how well the model did in predicting benign requests correctly by measuring the number of false classifications of benign requests.

Table 6 Performance results from training and testing the DNN model with the CSIC 2010 dataset

Experiment	Precision (%)	Recall	F-measure	FPR	Accuracy (%)
On 10676 requests (malicious and benign)	86.66	0.86	0.87	0.164	85.17

Table 7 Performance results of the CNN model trained with the CSIC 2010 dataset and tested with 43115 requests from the ECML/PKDD 2007 dataset

Precision (%)	Recall	F-measure	FPR	Accuracy (%)
36.01	0.91	0.52	0.79	43.86

Table 8 Performance results of the DNN model trained with the CSIC 2010 dataset and tested with 43115 requests from the ECML/PKDD 2007 dataset

Precision (%)	Recall	F-measure	FPR	Accuracy (%)
32.86	1.00	0.49	0.998	32.95

4.3.2 Performance Results with the ECML/PKDD 2007 Dataset

The dataset was divided with 70% (30180) of the dataset used for training and 30% (12935) used for validation. The CNN model achieved training accuracy of 94.73% and validation accuracy of 94.74%, the DNN model achieved training accuracy of 91.89% and validation accuracy of 91.90%. The results of the various experiments carried out on this dataset are shown in Tables 9–14, 21 and 22.

Table 9 Performance results of the CNN model trained and tested with the ECML/PKDD 2007 dataset

Experiment	Precision (%)	Recall	F-measure	FPR	Accuracy (%)
On 12935 requests (malicious and benign)	100.00	0.84	0.91	0.00	94.70

Table 10 Performance results on the CNN model trained on the ECML/PKDD 2007 dataset and tested with various types of requests (benign and malicious) present in the ECML/PKDD 2007 dataset

Request Type	Recall	FPR	Accuracy (%)
Benign	–	0.00	100.00
SSI Injection	0.83	–	83.27
SQL Injection	0.82	–	82.44
XPath Injection	0.84	–	83.75
Path Traversal	0.84	–	83.91
Command Injection	0.89	–	89.11
LDAP Injection	0.84	–	83.56
XSS	0.83	–	83.39

Table 11 Performance results of the DNN model trained and tested with the ECML/PKDD 2007 dataset

Experiment	Precision (%)	Recall	F-measure	FPR	Accuracy (%)
On 12935 requests (malicious and benign)	98.03	0.77	0.87	0.008	92.09

Table 12 Performance results on the DNN model trained on the ECML/PKDD 2007 dataset and tested with various types of requests (benign and malicious) present in the ECML/PKDD 2007 dataset

Request Type	Recall	FPR	Accuracy (%)
Benign	–	0.008	99.23
SSI Injection	0.81	–	81.29
SQL Injection	0.76	–	76.50
XPath Injection	0.84	–	83.53
Path Traversal	0.59	–	59.20
Command Injection	0.85	–	84.69
LDAP Injection	0.77	–	77.01
XSS	0.83	–	82.98

Table 13 Performance results of the CNN model trained with the ECML/PKDD 2007 dataset and tested with requests from the CSIC 2010 dataset

Precision (%)	Recall	F-measure	FPR	Accuracy (%)
84.42	0.34	0.48	0.076	60.01

Table 14 Performance results of the DNN model trained with the ECML/PKDD 2007 dataset and tested with requests from the CSIC 2010 dataset

Precision (%)	Recall	F-measure	FPR	Accuracy (%)
96.94	0.26	0.41	0.01	58.73

4.3.3 Performance Results with the Hybrid 2020 Dataset

The Hybrid 2020 dataset was divided with 70% (40184) used for training and 30% (17222) used for validation. The CNN model achieved a training accuracy of 95.59% and a validation accuracy of 95.95%, the DNN model achieved a training accuracy of 92.19% and a validation accuracy of 92.67%. The results of the various experiments carried out on this dataset are shown in Tables 15–20.

Table 15 Performance results of the CNN model trained and tested with the Hybrid 2020 dataset

Experiment	Precision (%)	Recall	F-measure	FPR	Accuracy (%)
On 17222 requests (Benign and Malicious)	99.71	0.92	0.96	0.003	95.95

Table 16 Performance results of the DNN model trained and tested with the Hybrid 2020 dataset

Experiment	Precision (%)	Recall	F-measure	FPR	Accuracy (%)
On 17222 requests (Benign and Malicious)	98.21	0.87	0.92	0.02	92.67

Table 17 Performance results of the CNN model trained with the Hybrid 2020 dataset and tested on the CSIC 2010 dataset

Precision (%)	Recall	F-measure	FPR	Accuracy (%)
70.95	0.39	0.5	0.19	57.59

Table 18 Performance results of the DNN model trained with the Hybrid 2020 dataset and tested on the CSIC 2010 dataset

Precision (%)	Recall	F-measure	FPR	Accuracy (%)
54.81	0.9	0.68	0.9	53.61

Table 19 Performance results of the CNN model trained with Hybrid 2020 dataset and tested separately on benign and malicious requests of the CSIC 2010 dataset

Experiment	Recall	FPR	Accuracy (%)
Benign samples	–	0.271	72.90
Malicious samples	0.44	–	44.14

Table 20 Performance results of the DNN model trained with Hybrid 2020 dataset and tested separately on benign and malicious requests of the CSIC 2010 dataset

Experiment	Recall	FPR	Accuracy (%)
Benign samples	–	0.990	1
Malicious samples	0.93	–	93.22

Table 21 Performance results of the CNN model trained with the ECML/PKDD 2007 dataset and tested separately on benign and malicious requests of the CSIC 2010 dataset

Experiment	Recall	FPR	Accuracy (%)
Benign samples	–	0.027	97.28
Malicious samples	0.18	–	18.17

Table 22 Performance results of the DNN model trained with the ECML/PKDD 2007 dataset and tested separately on benign and malicious requests of the CSIC 2010 dataset

Experiment	Recall	FPR	Accuracy (%)
Benign samples	–	0.009	99.13
Malicious samples	0.25	–	25.17

4.4 Discussion of Results

From the Figures 3–5, it is shown that the CNN model performs better than the DNN model on all the three datasets. The CNN model and DNN model trained with the ECML/PKDD 2007 and Hybrid 2020 datasets showed better performance when tested with the CSIC 2010 dataset as compared to training them on CSIC 2010 dataset and testing with ECML/PKDD 2007 and Hybrid 2020 datasets (cf. Tables 7, 8, 13, 14, 17, and 18). This is because the way each dataset was compiled and classified by experts differ greatly; the CSIC 2010 dataset was compiled based on an e-commerce web application and has a lot of reoccurring request parameters classified as benign and malicious depending on the URL or HTTP method with which the request parameter is sent. Another factor is that the CSIC dataset does not cover as much injection attacks as the ECML/PKDD 2007 and the Hybrid 2020 datasets. One other reason for the differing performance among the datasets is the embedding table in the embedding layer. The embedding table is optimized during training so the representations of characters of the models of the three different datasets are not completely similar. Because there is no existing pre-trained character embedding table, the embedding layer in the models have to optimize the randomly initialized embedding table based on the inputs given to the network which may make the models not to be able to generalize well outside their current dataset.

As stated earlier, the CNN model's performance was better than the DNN model's performance. The DNN had a lower performance due to factors like limited computational resources, the width and depth of the neural network and the compilation of the dataset. The DNN model required more computational resources as training was performed with over 15 million parameters as

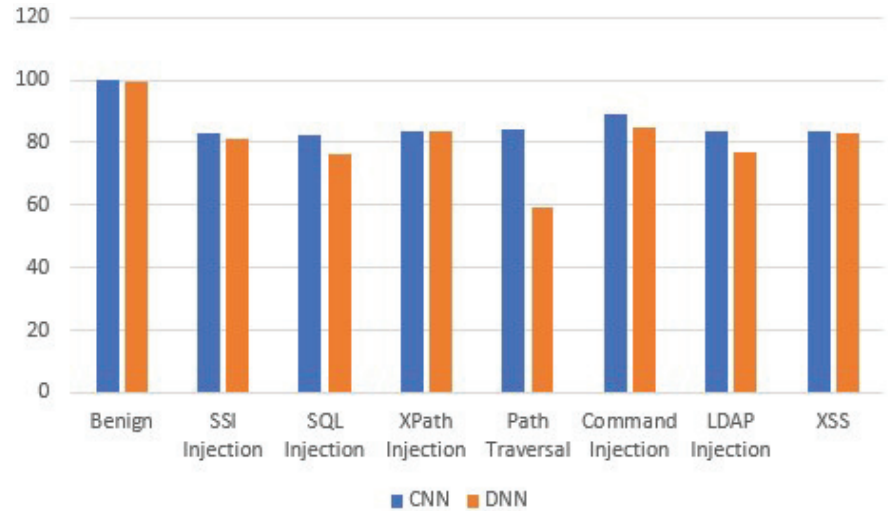

Figure 3 Comparison of accuracy of the DNN and CNN models trained on the ECML/PKDD 2007 dataset and tested on various types of requests (benign and malicious) present in the ECML/PKDD 2007 dataset.

compared to the CNN training with just 9 million parameters. The increase in parameters came from the flattening of the embedding layer and due to this, the DNN could not go much deeper or wider as that would exponentially increase the number of trainable parameters and the available computational resources would not be able to cater for such network.

Despite the limitations, the current architecture of the DNN model is well enough to classify even better if the datasets have provided more examples of HTTP injection attacks and benign requests in order to adjust the weights of the network to a more optimal value. The CNN performance is attributed to the advantages of a convolutional neural network, which are convolutions and parameter sharing and these allow it to achieve better abstraction than the DNN model.

Results from training the models with the ECML/PKDD 2007 dataset show a recall not less than 0.77 and a maximum FPR of 0.008 as seen in Tables 9 and 11. Results from training the models with the CSIC 2010 dataset show a recall not less than 0.86 and a maximum FPR of 0.164 as shown in Tables 5 and 6. When the models trained with the CSIC 2010 dataset were evaluated using the ECML/PKDD 2007 dataset, it was observed that the models performed poorly. This could be because the CSIC 2010 dataset was

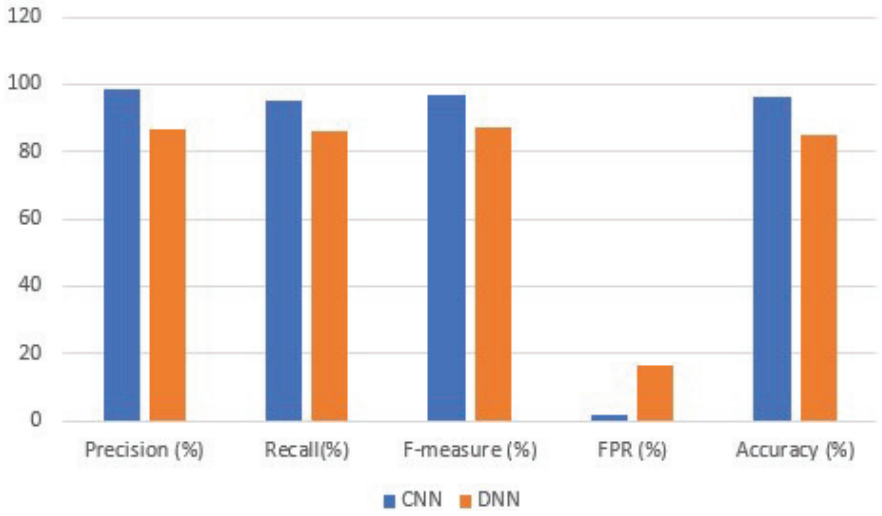

Figure 4 Performance comparison of the CNN and DNN models trained on the CSIC 2010 dataset and tested on the CSIC 2010 dataset.

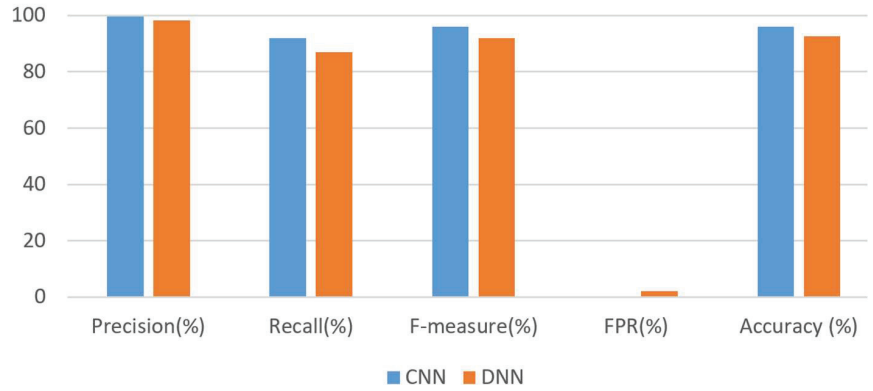

Figure 5 Performance comparison of the CNN and DNN models trained on the Hybrid 2020 dataset and tested on the Hybrid 2020 dataset.

not compiled to cater for cases the ECML/PKDD 2007 dataset was compiled for (cf. Figure 6).

We also experimented testing the trained models separately with malicious requests and separately with benign requests as shown in Tables 21 and 22. When evaluated with the CISC 2010 dataset, the ECML/PKDD 2007 dataset trained models only performed better in classifying benign

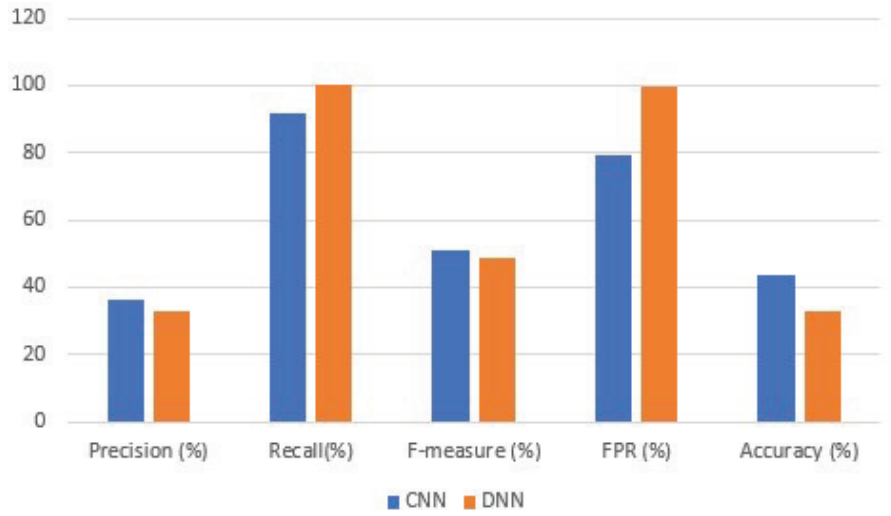

Figure 6 Performance comparison of the CNN and DNN models trained with the CSIC 2010 dataset and tested on the ECML/PKDD 2007 dataset.

requests but performed not so well in identifying malicious requests (cf. Figure 7) which was due to the compilation of the CSIC 2010 dataset as earlier explained. It can also be observed that the performance of the two models when tested on the benign requests has only a very slight difference.

The introduction of the PayloadsAllTheThings dataset to serve as a means to oversample the malicious examples in the ECML/PKDD dataset in order to achieve a balance on the distribution of malicious and benign samples in the dataset (resulting in a new dataset we referred to as Hybrid 2020 dataset) brought about some interesting observations as seen in Tables 15 and 16. The CNN still has the upper ground in performance matrices as expected based on earlier discussions. Taking the trained model and testing it with the CSIC dataset separately on benign requests and separately on malicious requests brings about new observations (Tables 19 and 20); the CNN is somewhat good on benign samples and does poorly on malicious samples while the DNN does poorly on the benign samples and does so well on malicious samples. One of the possible reasons why these observation occur lies in the level of abstraction of each model. The DNN when faced with malicious requests simulates a pattern recogniser because it needs to identify a character or phrase it considers malicious from it's training to classify incoming requests as malicious, this is due to the way the DNN

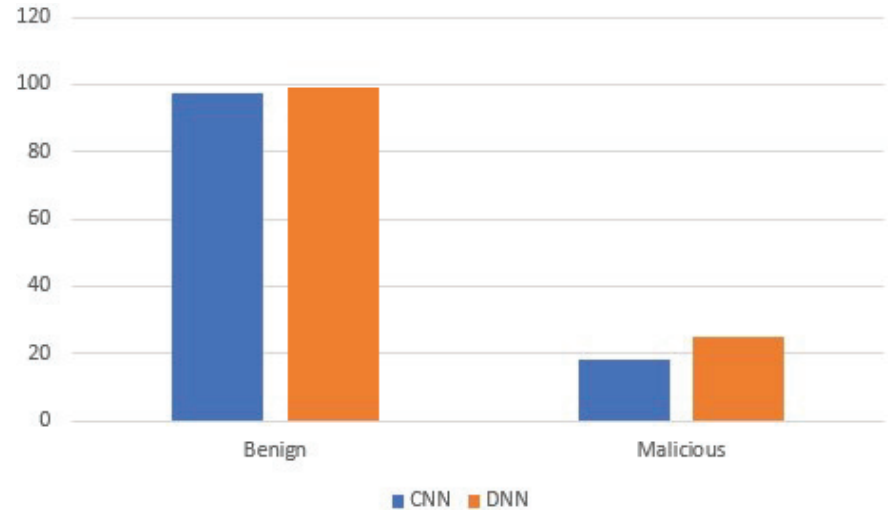

Figure 7 Accuracy Performance of the CNN and DNN model trained with the ECML/PKDD 2007 dataset and tested separately on benign and malicious requests from the CSIC 2010 dataset.

creates it's abstract representations of the requests. However CNN creates higher abstractions compared to the DNN and then classify based on the abstractions, the CNN model is able to work better in identifying benign because it is easy to get a general abstraction for benign requests in the Hybrid 2020 dataset that can recognize benign requests in the CSIC 2010 dataset. This is because CNN is able to reduce noise found in data samples via its series of convolutions. The DNN however is weak on the benign samples because the DNN abstractions are noisy compared to the CNN, Also because there isn't a standard embedding table for HTTP requests like there are for sentiment problems, it would be difficult for the DNN trained on different datasets to generalise benign representations to other datasets accurately, same is the case for the CNN on malicious requests.

A general look at the results of the models when trained with a dataset and tested on another shows that the CNN does better in identifying the benign requests and the DNN does better when identifying malicious requests and as explained earlier with the results concerning the Hybrid 2020 dataset, this is due to the way the CNN and DNN models generate their abstract representations. The CNN, through convolutions, is able to reduce the noise in the data which makes it easier to identify benign requests even on datasets it was not originally trained with, but with malicious requests it is dependent

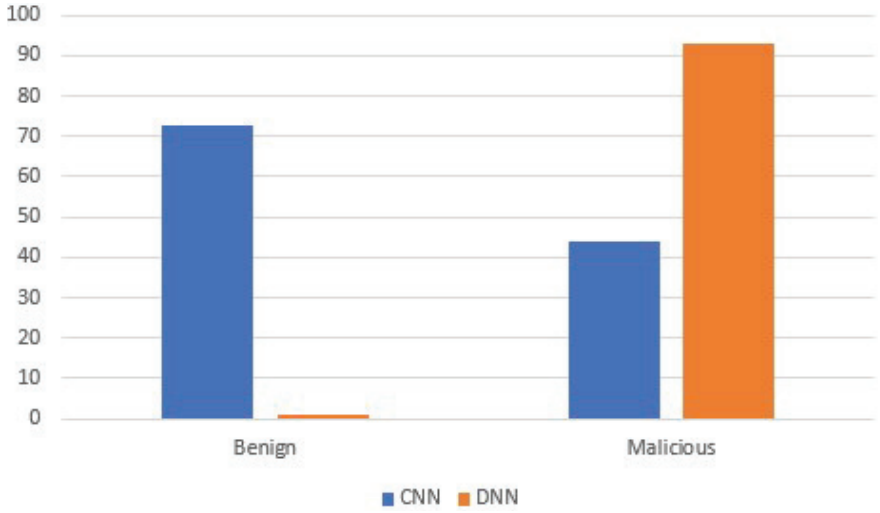

Figure 8 Accuracy Performance comparison of the CNN and DNN models trained with the Hybrid 2020 dataset and tested separately on benign and malicious requests from the CSIC 2010 dataset.

on the dataset used because malicious requests can differ based on the web application involved. The DNN on the other hand works like a regular expression equation looking for a pattern or character it can easily use to identify malicious requests, it flags any request that contains any character it deemed malicious during training so it can be ported to another dataset and do relatively well or even better if more samples are given. The issue the DNN has with identifying benign requests in other datasets is as a result of the limitations in the building of the DNN due to limited computational resources and the fact that there is no general character embedding for HTTP requests. It is also affected by the nature of the dataset it was trained with. A good recommendation would be to build an ensemble of the deep learning models in order to take advantage of both models on each type of request especially when testing on data it was not trained with.

In general, the results show that deep learning (DL) models are able to perfectly generate higher abstract representations of an HTTP request parameter and also able to correctly classify the requests. If trained on good datasets, DL models would be able to classify HTTP requests correctly for any web application and also an ensemble of DL models would lead to even better results because in this study there are cases where the DNN cover for the CNN and vise-versa (Figures 6–8).

5 Summary and Conclusion

This study focused on automatic detection of injection attack in HTTP requests. The study first began with a review of HTTP requests, HTTP injection attacks and deep learning. To give better insight on the project study, related works done in HTTP attack detection were reviewed. Deep learning techniques were used to develop models that would automatically detect injection attacks in HTTP requests. A special layer called the character embedding layer is used to allow the learning of the representation of the request parameter of HTTP requests in higher abstract levels and learning the relationships between the characters of the request parameter. The experimentation results showed that with deep learning, better injection attack detection is possible and given the right dataset, a trained deep learning detection model would be able to correctly classify for any web application.

In conclusion, HTTP injection attacks are dangerous to web servers and affect the safety of the Internet at large and better detection models are needed in order to automatically detect such attacks. This study took the approach of deep learning techniques to develop detection models capable of identifying injection attacks before the server acts on the requests, based on just the request parameters alone and was able to get good accuracy (not less than 85% for both models tested on the validation set of each dataset's training set). The CNN model outperformed the DNN model in various test experiments.

References

[1] R. Fielding and R. Julian, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC Editor, 2014.

[2] R. M. Lomte and S. A. Bhura, "Survey of different Web Application Attacks and its Preventive Measures," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 14, no. 5, pp. 46–51, 2013.

[3] E. Shafie, "Runtime Detection and Prevention for Structure Query Language Injection Attacks," Leicester, 2013.

[4] I. Goodfellow, Y. Bengio and A. Courville, Deep learning, Massachusetts: MIT press, 2016.

[5] Y. LeCun , Y. Bengio and G. Hinton, "Deep Learning," *Nature*, pp. 436-444, 2015.

[6] B. Gallagher and T. Eliassi-Rad, "Classification of HTTP attacks: A study on the ECML/PKDD 2007 discovery challenge," Lawrence Livermore National Lab, Livermore, 2009.

[7] A. S. Choudhary and M. L. Dhore, "CIDT: Detection of malicious code injection attacks on web application," International Journal of Computer Applications, vol. 50, no. 2, pp. 52–60, 2012.

[8] H. Lampesberger, P. Winter, M. Zeilinger and E. Hermann, "An Online Learning Stastical Model to Detect Malicious Web Requests," in *Security and Privacy in Communication Networks. SecureComm 2011. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 96, Berlin, Heidelberg, Springer, 2012, pp. 19–38.

[9] R. Kozik, M. Choras, R. Renk and W. Holubowicz, "A Proposal of algorithm for web applications cyber attack detection," in *IFIP International Conference on Computer Information Systems and Industrial Management*, Berlin, Heidelberg, 2015.

[10] M. B. Seyyar, F. Ö. Çatak and E. Gül, "Detection of attack-targeted scans from the Apache HTTP Server access," *Applied Computing and Informatics*, vol. 14, no. 1, pp. 28–36, 2017.

[11] Y. Dong and Y. Zhang, "Adaptively Detecting Malicious Queries in Web Attacks," arXiv preprint arXiv:1701.07774, 2017.

[12] S. Althubiti, X. Yuan and A. Esterline, "Analyzing HTTP requests for web intrusion detection," in *KSU Proceedings on Cybersecurity Education, Research and Practice*, Kennesaw, 2017.

[13] W. Rong, B. Zhang and X. Lv, "Malicious Web Request Detection Using Character-level CNN," in *Machine Learning for Cyber Security. ML4CS 2019. Lecture Notes in Computer Science*, vol. 11806, Springer, Cham, 2018.

[14] Swisskyrepo, "GitHub," 5 July 2019. [Online]. Available: http:/github.com/swisskyrepo/PayloadsAllTheThings. [Accessed 28 July 2020].

[15] F. Chollet, "Introducing keras 1.0.," 15 March 2015. [Online]. Available: https://blog.keras.io/introducing-keras-10.html.

[16] F. Chollet, Deep learning with Python, 1st ed., New York: Manning Publications, 2018.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V.

Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogenous Systems*, arXiv preprint arXiv:1603.04467, 2016.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[19] W. McKinney, "Data Structures for Statistical Computing in Python," in *Python in Science Conference*, 2010.

[20] S. van der Walt, C. S. Colbert and G. Varoquax, "The Numpy Array: A structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, pp. 22–30, 2011.

[21] T. Heitz and M. Roche, "Attack Challenge - ECML/PKDD Workshop," 2007. [Online]. Available: http://www.lirmm.fr/pkdd2007-challenge/index.html. [Accessed June 2019].

[22] C. T. Gimenez, A. P. Villegas and G. A. Maranon, "HTTP DATASET CSIC 2010," 20 January 2012. [Online]. Available: http://www.isi.csic.es/dataset.

Biographies

Victor Odumuyiwa obtained his PhD in 2010 from Nancy 2 University in France. He was an ETT fellow at the Massachusetts Institute of Technology, Boston, USA in 2013. He is a certified security intelligence analyst and application security analyst. He is also a member of the Global EPIC eco-system centred on innovation and cyber security. He is the Director of the Centre for Information Technology and Systems (CITS) and a Senior Lecturer at the Department of Computer Science, University of Lagos, Nigeria.

Analogbei Chibueze is a graduate of computer science from the University of Lagos, Nigeria with the passion for research and development of critical problem solving software solutions. He's research areas include artificial intelligence, deep learning, and modelling and simulation.