DETECTING SQL INJECTION AND XSS ATTACK

MSc Research Project
MSc in Cybersecurity

Nitin Vijla
Student ID: 22183981

School of Computing
National College of Ireland

Supervisor : Mr. Michael Pantridge

National College of Ireland

National College of Ireland

Project Submission Sheet

Student Name: Nitin Vijla

Student ID: 22183981

Programme: Msc in Cybersecurity **Year:** 2023-24

Module: Thesis

Lecturer: Michael Pantridge

Submission Due Date: 12/08/2024

Project Title: Detecting SQL Injection and XSS Attack

Word Count: .....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Nitin Vijla

Date: 12/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI Acknowledgement Supplement

[Insert Module Name]

[Insert Title of your assignment]

Your Name/Student Number	Course	Date
22183981	Msc Cybersecurity	12/08/2024

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

AI Acknowledgment

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used**.

[Insert Tool Name]	
[Insert Description of use] **- N/A**	
[Insert Sample prompt] **- N/A**	[Insert Sample response] - **N/A**

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:

N/A

Additional Evidence:

N/A

Abstract

This study aims to enhance the identification of XSS and SQL injection as threats by developing a CNN-LSTM model. XSS and SQL injection threats remain apparent when it comes to the security of web applications. Accordingly, for enhancing the detection accuracy and the model's ability to maintain invariance, CNNs are adopted for performing local feature extraction from the input sequences and LSTMs for modeling sequential relationships. As a result, the study works out and compares the model most notably with the tests on the accuracy and the real-time responses. The study shows that these techniques have significant improvements in recognizing such assaults in contrast to the conventional approaches, that is, devising a more reliable method to protect the web-based applications against new and developing threats from hackers.

Keywords: CNN, RNN, XSS, SQL, LSTMs

TABLE OF CONTENTS

Introduction

This study addresses the vital issue of cybersecurity by focusing on the identification of the SQL injection and cross-site scripting (XSS) attacks through the state of the art machine learning methods. Web applications are normally under threat from SQL injection as well as XSS, which makes it possible for a website to have security errors and leak data. Because of the constantly evolving and complex nature of these procedures, the conventional detection tools do not work most of the time. Thus, the attempt of this paper is to develop a new model that holds benefits of both LSTM and convolutional networks for the purpose of improving the basic detection effectiveness. The proposed model aims at enhancing the detection performance by incorporating LSTM's capability in capturing chronological relations and CNN's capacity in identifying local patterns. Based on the work of Zhang et al. , 2022 and Krishnan et al. , 2021, this work integrates recent advancements in deep learning to present an effective solution toward combating SQL Injection and XSS attacks.

Research Questions

The research questions for this study are as following:

1. How different hyperparameters influence the performance of the proposed hybrid CNN-RNN model?
2. What distinguishes non-attack queries from XSS attack queries in view of textual characteristics?
3. How effective are the different evaluation criteria in measuring the efficacy of the model in detecting cross-site scripting attacks?

The recommended proposed solution is the development of a CNN-RNN model that will be able to recognize XSS and SQL injection attacks in real time. This model is an improvement over normal techniques because by combining the advantages of both CNNs and RNNs in the proposed system, the detection accuracy can be higher by considering sequential dependence and local information in the query strings.

The use of hybrid CNN-RNN has the following main benefits for the detection of XSS and SQL injection attacks.

- Better Feature Extraction: As a result, the CNN component of the hierarchy is beneficial because it can potentially reduce how much human feature engineering is necessary by extracting suitable local features from the input query automatically. It has the capability of detecting small patterns that may indicate intrusions that traditional rule-based systems may consider.
- Understanding Sequential Context: The RNN (LSTM) part allows the model to work with them in consideration with the sequential significance of the series of characters in a query allowing the model to identify the context and purpose of potentially malicious inputs.
- Adaptability: This deep learning approach is less sensitive to altering threats since, unlike other rule-based systems, it can retrain a new look for novel dangers' networks.

- Real-time Detection: In the training phase, once the algorithms in the model are learned, the model provides real-time capabilities of possible intrusions in working environments.
- Decreased False Positives: In comparison with the other kinds of detection approaches, the model may use complex patterns learned from a large number of samples and thus have fewer false positive results. This would lead to a reduction of the number of false alarms as well as the unwanted interferences.
- Interpretability: Through techniques like attention, it is possible to gain more insight into what makes certain queries an assault, which, with the help of feature visualization learning, can aid in enhancing security models.

Literature Review

A. Deep Learning Techniques for SQL Injection Detection

According to Zhang et al. (2022), database security can be effectively defended by identifying SQL injection attacks through a deep neural network system known as SQLNN. Their work involves the adoption of a multi-hidden layer neural network with ReLU activation, the construction of a sparse matrix, and the word vectorization of SQL statements. To overcome this issue and enhance the ability of the model to generalize, the model incorporates the Dropout method and an optimal loss function. As mentioned by Zhang et al. [], its overall identifying accuracy is higher than 96% and, as for other indicators such as the accuracy, precision, recall, and F value, it is more efficient than LSTM and traditional machine learning algorithms. They emphasize that their model addresses the issue of overfitting and removes the need for customary feature engineering. In contrast, Krishnan et al. (2021) employed a variety of machine learning methodologies including Naive Bayes, CNN, SVM, Passive Aggressive Classifier as well as Logistic Regression for the purpose of detecting an SQL injection. Based on their work, CNN was found to be effective when compared to any of the other algorithms and had achieved accuracy of 97% in detecting the SQL injections.

As noted by Krishnan et al. , machine learning is crucial in cybersecurity since existing signature-based models serve less effectively against constantly evolving and diverse attack patterns. CNN stands as a strong asset in both situations, thereby supporting the effectiveness of deep learning approaches in SQL injection detection. While Krishnan et al. have provided a detailed analysis of several machine learning algorithms, the SQLNN model that has been proposed by Zhang et al. appears to be a more robust method for detecting the SQL injection type of attacks. These two papers highlight the need to come up with learning-based and adaptive mechanisms in order to mitigate against sql injection attacks given their evolutionary characteristic.

B. Machine Learning Techniques for Detecting XSS Attacks.

Kaur et al. (2023) have identified that Cross-Site Scripting (XSS) attacks remain an important threat to web application security, which is still listed among the ten most dangerous vulnerabilities mentioned by OWASP. By breaking down the core findings of the study, they stress how machine learning and neural network approaches are becoming increasingly

important for cross-site scripting (XSS) assault detection. To this end the authors provide a review of present day detection methods including deep neural networks, decision trees as well as web-log-based models and emphasize on fresh approaches since the attack techniques are growing to be more and more superior.

On the other hand, Habibi and Surantha (2020) focus on evaluating specific ML approaches that can be used in identifying the XSS type of attacks. The Naïve Bayes (NB), K-Nearest Neighbor (KNN), and Support Vector Machine (SVM) classifiers are under analysis in this study in order to determine which classifier yields the best results. In their algorithms, the authors have incorporated the n-gram approach to get more features from the script text. From the tests they conduct, it can be seen that the highest level of accuracy of 98% is achieved by integrating SVM and n-gram in the detection of XSS attack.

Combined with these papers, one can observe that while XSS attacks remain a major threat, more and more research is carried out to develop better methods and systems for XSS detection, which are less prone to false positives and more accurate; machine learning methods can be very useful in improving the protection of web applications.

C. Conclusion of Reviewed Works

This literature review provides valuable information about the current research in the area of using machine learning and deep learning algorithms in the detection of SQL injection and XSS attack. It shows how effectively deep neural networks, especially the CNN-based model, can detect such attacks. As demonstrated in the analysis, these sophisticated tactics outcompete traditional methodologies and can adapt to shifts in an attack's patterns. The development of research in this area can be observed in the comparison with other studies which shift from comparing various machine learning algorithms to designing dedicated models such as SQLNN. Furthermore, it underlines how important feature extraction techniques are to increase the detection rates, such as word vectors, and the n-gram techniques. The present evaluation helps to define possible directions for further research, for example, the application of various approaches and the improvement of deep learning models. It also points at the fact that only adaptive, learning-based approaches are needed to address the problem because web application attacks are constantly evolving.

Research Method

A. Research Method

The aim of this project is to develop and evaluate an effective model using Recurrent Neural Networks and Convolutional Neural Networks to detect two main threats of SQL injection and Cross-Site Scripting (XSS). Herein, an attempt is made at enhancing the accuracy of the existing detection system through the supplementation and incorporation of CNN and RNN structures. In this way, the research aims to address the limitations of the prevailing methodologies and enhance the recovery rates achieved by machine learning algorithms. The study will also seek to determine how the design of the model can help minimize false positives while maintaining a high detection rate Nair (2024) . The researcher expects to advance the knowledge of the current existing plans of attack detection using deep learning approaches by conducting extensive testing

and evaluating them for effectiveness in the security of web applications Bhardwaj et al. (2022) . The goal is to deliver practical and realistic solutions that will enhance the security of web applications from XSS and SQL injection, which may be leading to enhanced Internet Security for both the common individuals and the corporate entities.

B. Planned Steps

Literature Review and Model Design

This study has provided a detailed literature review of the recent research on the detection of SQL injection and XSS attack detection in order to understand and compare machine learning approaches and literature. The review observed that due to the capacity to detect complex features in the underlying input data, there is an increasing trend towards deep learning, with CNN and RNR models. Many approaches were discussed, for instance, comparative evaluation of several machine learning techniques presented by Krishnan et al. (2021) and SQLNN model introduced by Zhang et al. (2022). Such research helped to create the hybrid CNN-RNN model providing valuable information about the strengths and weaknesses of different strategies. Thus, in an attempt to combine the strengths of the architecture of the CNN and RNN, the hybrid model is employed. CNN component aimed at seeking hierarchical structures in the web scripts and SQL queries by extracting local features from input sequence. As it has already been mentioned, to consider the sequential dependencies in the data, the RNN part was included in the question, to be precise, LSTM layer. This made it possible for the model to take into account the context of whole queries or even scripts which might be an element of a larger one. The exactly proposed model includes LSTM layer to regulate the sequential flow of data and FC layer for classifying the last output, a 1D convolution layer to identify the features of desired sequence, a max pooling layer to minimize the dimension and an embedding layer which converts the tokenized input into the high dimension vector Qbea'h et al. (2020). . Different from single architecture models, this kind of hybrid method will allow for less probabilities of misclassifications of XSS and SQL injection threats and will increase the efficiency and accuracy of the detection.

Data Collection and Processing

All the primary data for the purpose of conducting the study to develop the model are sourced from Kaggle. Three datasets Train. csv, Test. csv, and Validation. Two csv files, one included questions labeled as benign/malicious were employed in this research study Alsaffar et al. (2022).These datasets provided a comprehensive preparation for training and evaluating the model because these databases were scrutinized to ensure the presence of various SQL injection and XSS attack types.

```
def create_subset(file_path, subset_size, chunk_size=10000):
    subset_list = []
    for chunk in pd.read_csv(file_path, encoding='utf-8', chunksize=chunk_size):
        subset_list.append(chunk.sample(frac=subset_size / chunk_size, random_state=42))
        if len(subset_list) * chunk_size >= subset_size:
            break
    return pd.concat(subset_list).reset_index(drop=True)

# Defining subset sizes
train_subset_size = 10000
test_subset_size = 5000
validation_subset_size = 5000

# Creating subsets
try:
    train_subset = create_subset('Train.csv', train_subset_size)
    test_subset = create_subset('Test.csv', test_subset_size)
    validation_subset = create_subset('Validation.csv', validation_subset_size)
except pd.errors.ParserError as e:
    print(f"Error creating subsets: {e}")

# Verifying the subset sizes
print(f"Train subset size: {len(train_subset)}")
print(f"Test subset size: {len(test_subset)}")
print(f"Validation subset size: {len(validation_subset)}")
```

```
Train subset size: 10000
Test subset size: 5000
```

Figure 1: Creating Subsets

(Source:-Self-Created in Jupyter Notebook)

To inspect the structure of the data and make sure encoding is correct, a very small subset of the data is initially loaded into the memory. Subsequently, in order to optimally control the computational resources, related subsets came up as follows Choudhary et al. (2021). Specifically, 10,000 samples were tested for the training set, 5,000 samples, for the test set and the balance 5,000 samples, for the validation set.

```
*[13]: from tensorflow.keras.preprocessing.text import Tokenizer
       from tensorflow.keras.preprocessing.sequence import pad_sequences
       import torch

       # Combining subsets for tokenization
       all_queries = pd.concat([train_subset['Query'], test_subset['Query'], validation_subset['Query']], axis=0)

       # Tokenizer and padding setup
       tokenizer = Tokenizer()
       tokenizer.fit_on_texts(all_queries)
       max_length = max([len(seq) for seq in tokenizer.texts_to_sequences(all_queries)])

       def preprocess_data(df, tokenizer, max_length):
           sequences = tokenizer.texts_to_sequences(df['Query'])
           padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post')
           labels = df['Label'].fillna(0).astype(int).values
           return padded_sequences, labels

       X_train, y_train = preprocess_data(train_subset, tokenizer, max_length)
       X_test, y_test = preprocess_data(test_subset, tokenizer, max_length)
       X_validation, y_validation = preprocess_data(validation_subset, tokenizer, max_length)
```

Figure 2: Tokenizing and padding the data

(Source:-Self-Created in Jupyter Notebook)

The process that followed was tokenization of the textual data. All the queries issued by the subgroups were fitted through a tokenizer thereby arriving at a vocabulary that maps words to different integers. Subsequently, the inquiries were turned into these integer sequences: Padding was applied to maintain the input size equal in order to meet the requirements of the convolutional and recurrent part of the model in order to address the issue connected with difference in length of queries Abikoye et al. (2020). It put the documents and their corresponding labels in sequences, then tokenized and padded them, followed by their conversion into tensors compatible with PyTorch used in this study.

```
*[14]: from torch.utils.data import DataLoader, TensorDataset

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Converting to tensors
X_train_tensor = torch.LongTensor(X_train).to(device)
y_train_tensor = torch.LongTensor(y_train).to(device)
X_test_tensor = torch.LongTensor(X_test).to(device)
y_test_tensor = torch.LongTensor(y_test).to(device)
X_validation_tensor = torch.LongTensor(X_validation).to(device)
y_validation_tensor = torch.LongTensor(y_validation).to(device)

# Creating DataLoader
train_data = TensorDataset(X_train_tensor, y_train_tensor)
test_data = TensorDataset(X_test_tensor, y_test_tensor)
validation_data = TensorDataset(X_validation_tensor, y_validation_tensor)

train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
test_loader = DataLoader(test_data, batch_size=32, shuffle=False)
validation_loader = DataLoader(validation_data, batch_size=32, shuffle=False)
```

Figure 3: Putting the tensors to data loaders

(Source:-Self-Created in Jupyter Notebook)

These tensors were then put in various forms of DataLoader to facilitate batch processing when it came to training and evaluation. Other extraction methods, which are also used to supplement the data set, are also considered in an attempt to improve increases it to the generality of the model, the addition of random noise is used. While before feeding it into the model, the data was preprocessed so that the model would go through severally attacks vectors and range of questions, as conducted by Muslihi & Alghazzawi, et al. (2020), such methods were undertaken effectively.

Model Implementation and Training

The preparation steps for the CNN-RNN model and training were important procedures that allowed moving forward to the construction of a constructive solution for XSS and SQL-injection threats detection. All inputs, which were in token form, passed through an embedding layer of the model which also incorporated semantics of the tokens Muslihi and Alghazzawi, (2020). After the embedding, a one-dimensional convolutional layer was used. layer to extract the features of the local regions from the input sequences.

```
*[15]: import torch.nn as nn

class CNNRNNModel(nn.Module):
    def __init__(self, vocab_size, embed_size, num_classes):
        super(CNNRNNModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.conv = nn.Conv1d(in_channels=embed_size, out_channels=128, kernel_size=5)
        self.pool = nn.MaxPool1d(kernel_size=2)
        self.lstm = nn.LSTM(input_size=128, hidden_size=128, num_layers=2, batch_first=True)
        self.fc = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.embedding(x).permute(0, 2, 1)
        x = self.pool(torch.relu(self.conv(x))).permute(0, 2, 1)
        x, _ = self.lstm(x)
        x = self.fc(x[:, -1, :])
        return x

# Instantiating model
vocab_size = len(tokenizer.word_index) + 1
embed_size = 100
num_classes = 2
model = CNNRNNModel(vocab_size, embed_size, num_classes).to(device)
```

Figure 4: Integrating max pool layers with convolution layers

(Source:-Self-Created in Jupyter Notebook)

A max-pooling layer followed the convolutional layer to down-sample the feature maps, reducing dimensionality while highlighting the most features. Subsequently, an LSTM layer was used to extract the context and sequential information from the queries from the processed

features Farea et al. (2021). Due to its ability to handle long term dependencies which is essential for interpreting complex query syntactic structures and identifying the smart attack patterns the LSTM layer was used. Then, a dense layer with the input of the LSTM layer served to sort out the received queries as benign or malicious.

Figure 5: Training the model with adam optimizer

(Source:-Self-Created in Jupyter Notebook)

To build the model, cross-entropy loss which is used for binary classification was used and for optimizing the model, the Adam optimizer. To reduce the values of the loss function, and, therefore, achieve the goal of the model minimizing the loss function for driving chosen results, the model was trained over the training data for ten epochs with the batch size of thirty two. The gradients obtained from the loss at each epoch were used to update the values of the model's parameters Schoenborn and Althoff (2021). The epochs in which the loss was minimized were displayed and printed on screen to observe the training session loss. Different tests including validation were also performed at the end of every epoch in order to test the model's general applicability to unseen data. Therefore, during the training phase, possible over-fitting could be detected and save prospects of generalization Cui et al. (2020). During the process, training and implementation took place in a GPU-capable environment. This helped in faster calculation and hence made it possible to work with a vast dataset.

Testing and Model Optimization

Testing the effectiveness and efficiency of the proposed hybrid CNN-RNN model in identifying both XSS and SQL injection threats was done in this phase. Obviously, it was also important to validate the model using some other dataset not used in the training phase. With respect to this, perhaps the most common evaluation criterion was the identification accuracy, amounting to the percentage of effectively recognized inquiries as shown by Nasereddin et al. (2023). Other metrics used to give insight into the model performance were precision, recall, and F1, mainly in a comparison between benign and malicious queries. Testing the model: apply the trained model on the validation dataset to make the predictions. Then, calculate the performance measure, which gives a difference between these predictions and the true labels.

From the results obtained in this study, it can be noted that the model recorded a great accuracy in classifying a huge percentage of both benign and malicious queries. A closer look was taken

into the recall and precision measures to discover other areas that required enhancement, particularly on reducing the false positives and false negatives Tadhani et al. (2024). First, different numbers of layers, different batch sizes as well as different learning rates were tried and tested in a manner that aimed at finding out which combination enhanced the classification metrics and which reduced the loss function. Such a process was separated as hyperparameter tuning. To ensure the model's consistency when tested on several subsets of data, the cross-validation methods were also used.

```
*[19]: import time
       import random

       def preprocess_and_predict(query, model, tokenizer, max_length, device):
           sequence = tokenizer.texts_to_sequences([query])
           padded_sequence = pad_sequences(sequence, maxlen=max_length, padding='post')
           tensor = torch.LongTensor(padded_sequence).to(device)

           with torch.no_grad():
               output = model(tensor)
               _, predicted = torch.max(output, 1)

           return predicted.item()

       # Simulating a real-time stream of queries
       real_time_queries = [
           "SELECT * FROM users WHERE username = 'admin' --",
           "A pretentious but - to varying degrees - watchable...",
           "1 OR 1=1",
           "DROP TABLE users",
           "The movie was great! I loved the acting."

       ]

       for query in real_time_queries:
           prediction = preprocess_and_predict(query, model, tokenizer, max_length, device)
           print(f'Query: {query}')
           print(f'Prediction: {prediction}')
           print('-' * 50)
           time.sleep(random.uniform(0.5, 2.0))  # Simulating delay between queries
```

Figure 6: Testing the model with different queries

(Source:-Self-Created in Jupyter Notebook)

A better balance of the training set was achieved by making sure that as many searches were included with the primary intention of minimizing false positives. In addition, many of each type of question were produced by applying data augmentation techniques. To prevent overfitting of the model and increase the model's ability to generalize, the model also employed other forms of regularization such as dropout Rahul et al. (2021). To enhance the level of resistance additional types of ensemble methods, which include several models and their predictions, were considered. Altogether, these optimization works created a more reliable and efficient model that can find SQL injection and XSS threats with smaller amounts of false positives and false negatives.

Validation and Analysis

For the purpose of validation the outcomes of the model have to be tested in depth using a validation dataset that is different from the training dataset. This step was necessary to make the model accurate to other pieces of data that it has not been indoctrinated with before. In the case of the validation dataset, there were equal numbers of both the malicious and benign searches, and this made it possible to determine the capability of the model in identifying the two types of inputs Chen et al. (2021). Cross validation was performed in order to check the model and for every query in the validation set predictions were made.

```
[18]: from sklearn.metrics import accuracy_score, classification_report

# Validating the model on the validation set
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for X_batch, y_batch in tqdm(validation_loader, desc="Validating"):
        outputs = model(X_batch)
        _, predicted = torch.max(outputs, 1)
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(y_batch.cpu().numpy())

# Calculating accuracy
accuracy = accuracy_score(all_labels, all_preds)
print(f'Validation Accuracy: {accuracy}')
print(classification_report(all_labels, all_preds))
```

```
Validating: 100%|██████████| 157/157 [00:20<00:00,  7.65it/s]
Validation Accuracy: 0.97
```

Figure 7: Validating the model

(Source:-Self-Created in Jupyter Notebook)

After that, for calculating scores such as accuracy, precision, recall, and F1-score, these predictions were matched with the real labels. Precision and recall provided details about the exactness of the appropriate limitation of false positives and false negatives and actual recognition of real positives while accuracy tested the overall fitness level of the model Arock (2021). The F1-score simply offered one value that captured both the measures of precise and retrieved values since it is the harmonic mean of the two Sivasangari et al. (2021)

Tanakas et al. (2021).In the results section, the overall validation results on all criteria were represented, and it was illustrated how effectively the hybrid CNN-RNN model with excellent outcomes detected XSS and SQL injection attacks.

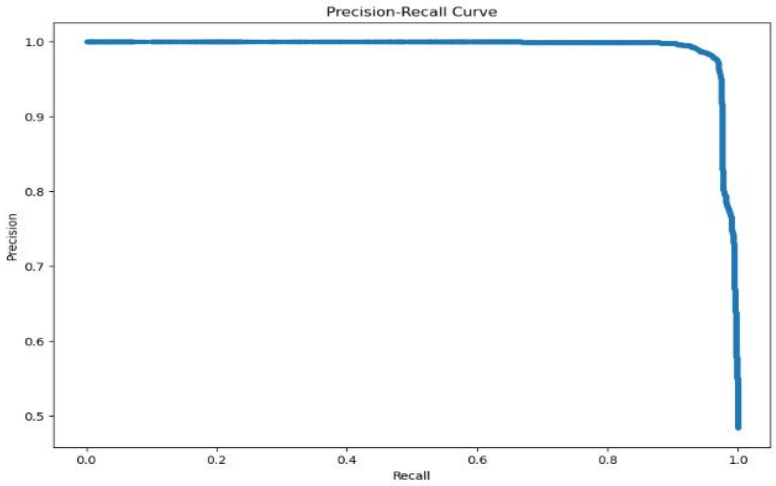


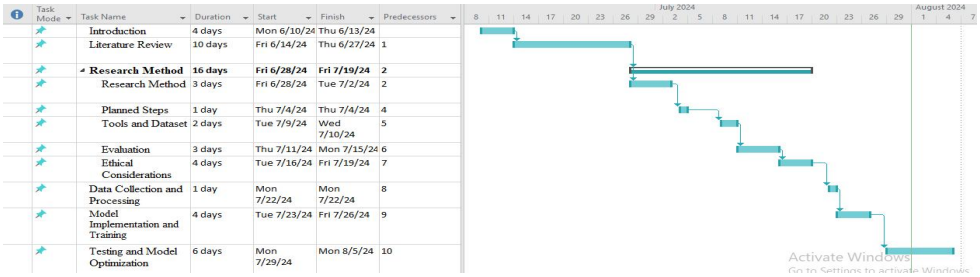
Figure 8: Displaying the precision recall curve

(Source:-Self-Created in Jupyter Notebook)

Further quantitative analysis of the metrics used in benchmarking the model and examining errors provided the detection rate and the false positive rate. The detection rate, also called recall, demonstrated the percentage of real attacks that the model recognised was a threat. This was done to ensure that the model could recognize security issues and thus the need for a high detection rate. On the other hand, the FPR determined the proportion of genuine calls that were presumed to be unsafe. This rate of false positives has to be kept as low as possible in order to avoid wastage of time in responding to unnecessary alarms and to maintain confidence in the reliability of the system. In order to clarify the strengths and weaknesses of the model, careful

analysis of predictions made in the validation dataset was taken into consideration. Applying different thresholds for the positive class, precision-recall curves could be plotted to get the details of the trade-off between recall and precision of the model at different probability thresholds Yadav and Kumar (2022). Moreover, the model was again tested using ROC (Receiver Operating Characteristic) curves based on which, the AUC (Area Under the Curve) was computed, determining the model's capability to distinguish between benign and malicious queries.

Project Timeline



Task Mode	Task Name	Duration	Start	Finish	Predecessors
	Introduction	4 days	Mon 6/10/24	Thu 6/13/24	
	Literature Review	10 days	Fri 6/14/24	Thu 6/27/24	1
	Research Method	**16 days**	**Fri 6/28/24**	**Fri 7/19/24**	2
	Research Method	3 days	Fri 6/28/24	Tue 7/2/24	2
	Planned Steps	1 day	Thu 7/4/24	Thu 7/4/24	4
	Tools and Dataset	2 days	Tue 7/9/24	Wed 7/10/24	5
	Evaluation	3 days	Thu 7/11/24	Mon 7/15/24	6
	Ethical Considerations	4 days	Tue 7/16/24	Fri 7/19/24	7
	Data Collection and Processing	1 day	Mon 7/22/24	Mon 7/22/24	8
	Model Implementation and Training	4 days	Tue 7/23/24	Fri 7/26/24	9
	Testing and Model Optimization	6 days	Mon 7/29/24	Mon 8/5/24	10

Figure 9: Gantt Chart
(Source: Self-Created)

C. Tools and Dataset

PyTorch: Due to PyTorch's flexibility and its provision for dynamic computation graphs it was employed to build and train the hybrid model. The idea of CNN-RNN architecture was quite easy to implement with the help of PyTorch, making tensor manipulation, enhancing the model parameters, and training through appropriate GPU, a lot easier.

Python: The choice of programming language for defining the model is based on the opportunity to support libraries and ease of usage of Python. Python was suited to meet data pretreatment, model formulation, and assessment responsibilities due to its codes' readability and flexibility. Moreover, the availability of numerous libraries that were vital for the particular project was provided by Python's environment.

TensorFlow/Keras: For tokenizing and padding, these preliminary steps involved in the text preprocessing, TensorFlow and particularly its deep learning API called Keras were used. Before feeding the text queries, the text queries were segmented and each text query was converted into a sequence of integers and all input sequences had to be of equal length, which was done using the pad sequences function, an important criteria when using batch processing in neural networks was used.

Scikit-Learn: Classification reports were created using Scikit-Learn along with performance metrics such as accuracy, precision, recall and F1 score measures. It integrated basic operations to check the model's performance on the validation set and analyze its results.

Pandas: Pandas served for the purpose of data analyzing and data manipulation. It became convenient in loading as well as analyzing the Train, Test and Validation datasets. csv datasets and proved to be beneficial in managing data during preprocessing and creating subsets.

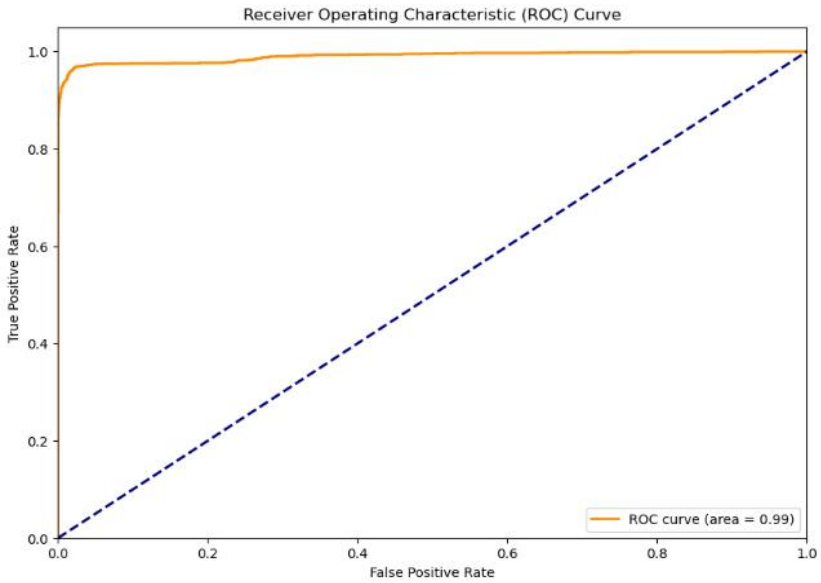


Figure 10: Displaying the ROC curve

(Source: Self-Created in Jupyter Notebook)

Charts and graphs as the model's performance was evaluated by using the visualization modules Matplotlib and Seaborn. Based on the flowchart, Matplotlib and Seaborn were used to draw the confusion matrices, curves of precision-recall, and ROC curves to show the behaviors of the model.

TQDM: The training and validation processes were also accompanied by TQDM which helped in tracking the process and visualizing it. It allowed tracking the model training progress and, therefore, facilitated the training process of the given model.

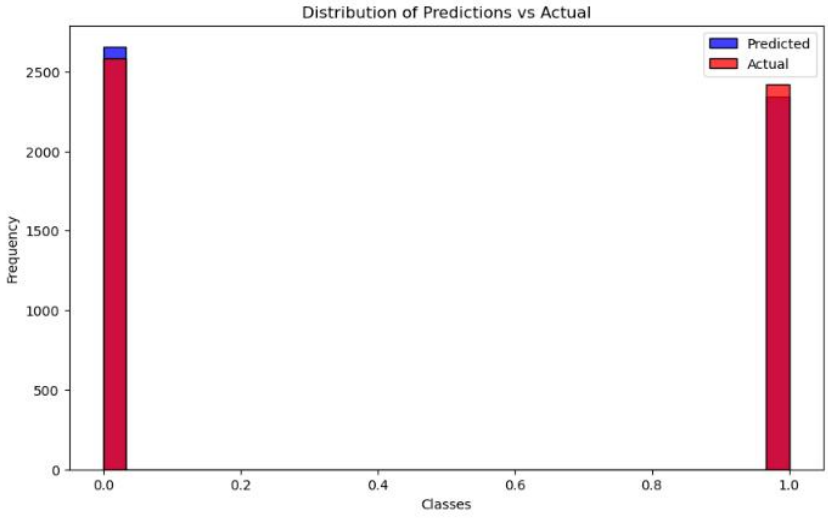


Figure 11: Displaying the distribution of predicted vs actual

(Source: Self-Created in Jupyter Notebook)

WordCloud: For visualizing distribution of words in the queries, help from WordCloud library was used. It generated word clouds for the benign as well as the toxic classes and provided information about the characteristics of different sorts of queries.

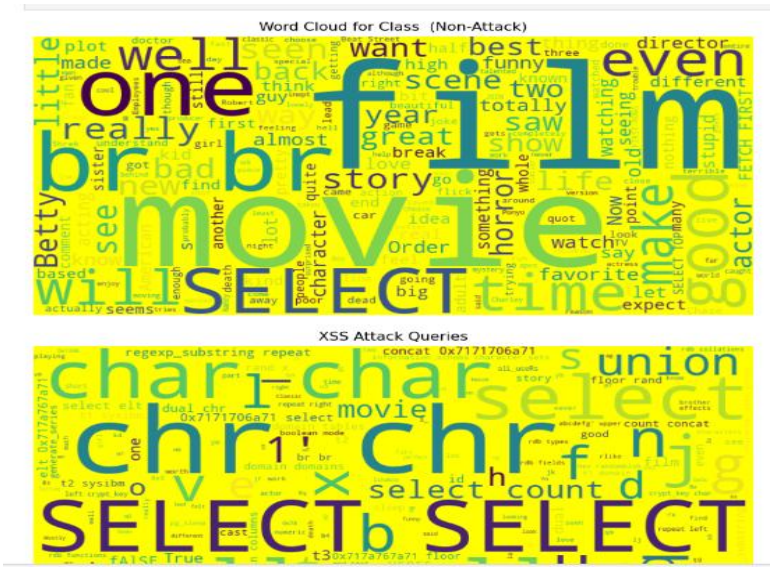


Figure 12: Displaying most common words as word cloud

(Source: Self-Created in Jupyter Notebook)

Therefore these tools and libraries have been used to construct and analyze the hybrid CNN-RNN model, which has proved its efficiency, thus indicating the capability of the model to detect SQL injection and XSS threats.

D. Evaluation

Accuracy:

Accuracy is one of the fundamental performance metrics which is used when developing classification models. It is calculated by establishing the ratio of total number of forecasts made to the marginal accuracy of the forecasts. The proposed hybrid CNN-RNN model revealed a spectacular validation accuracy of 97% in this investigation.

	precision	recall	f1-score	support
0	0.96	0.99	0.97	2581
1	0.98	0.95	0.97	2419
accuracy			0.97	5000
macro avg	0.97	0.97	0.97	5000
weighted avg	0.97	0.97	0.97	5000

Figure 13: Displaying the calculated classification metrics

(Source:-Self-Created in Jupyter Notebook)

Such high accuracy presupposes that the model is rather effective in classifying any given type of SQL injection and XSS attack queries as well as in distinguishing them from simple queries. The accuracy, calculated and reported with the help of simple functions provided by the Scikit-Learn toolkit, was used to define this statistic.

Realtime Performance:

To exemplify, in cybersecurity applications, the speed is crucial since it determines how the model performs given the threats of reality. To mimic the setting closely typical for a production system, namely real-time queries coming in at a constant rate, the model was tested with a constant stream of such queries. During this testing, the model's accuracy of the predicted values and its response time were recorded.

```
--------------------------------------------------
Query: A pretentious but - to varying degrees - watchable...
Prediction: 0
--------------------------------------------------
Query: 1 OR 1=1
Prediction: 1
--------------------------------------------------
Query: DROP TABLE users
Prediction: 0
--------------------------------------------------
```

Figure 14: Displaying the predicted outcomes by testing with different queries

(Source:-Self-Created in Jupyter Notebook)

The model was thought out to be able to peruse and sort the queries as fast as possible with any possible delay through the increase of the speed of the preprocessing and prediction phases. The use of PyTorch's GPU capability allowed for the hybrid CNN-RNN to perform satisfactorily as a real-time application while maintaining the highest level of accuracy and the lowest levels of latency Marashdeh et al. (2021). This becomes particularly useful in application when early identification of a threat would help to prevent future security incidents. Thus, the model demonstrates its suitability for performance as a reliable application in the dynamic and fast-moving environment for the detection of SQL injection and XSS attacks, where high precision is complemented by high real-time efficiency.

E. Ethical Considerations

The sources of possible bias in the data, in order to later generate prejudiced judgments, were assessed and accordingly dealt with. This would mean scrutiny of the information for any bias in the way that certain searches or assaults were presented, then using techniques such as oversampling or undersampling to correct such imbalances. It aimed to develop a model that is unbiased for decision making and would enhance model reliability in the detection of XSS and SQL injection attacks. Since data cleansing, model training, and evaluation are all documented, there is transparency. One can easily understand the method applied and even check the results, since the documentation ensures that the process is transparent and can be easily replicated. To facilitate this transparency training records of protocols, parameters, and assessment were kept.

Pertaining to data security and privacy, utmost importance was placed on the protection of the collected dataset. Measures were taken to ensure that the privacy of the data was upheld during processing and storage of data. In order to regulate the qualities of the presented mouths and prevent any security threats, of the given dataset, the access to the data was limited and in some scenarios when the procured regarding the targets of the involved identities, the data was anonymized.

Reference List

Journals

Zhang, W., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., & Zhi, G. 2022. Deep Neural Network-Based SQL Injection Detection Method. Security and Communication Networks, 2022(1), 4836289.

Krishnan, S. A., Sabu, A. N., Sajan, P. P., & Sreedeep, A. L. 2021. SQL injection detection using machine learning. Revista Geintec-Gestao Inovacao E Tecnologias, 11(3), 300-310.

Kaur, J., Garg, U., & Bathla, G. 2023. Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. Artificial Intelligence Review, 56(11), 12725-12769.

Habibi, G., & Surantha, N. 2020, August. XSS attack detection with machine learning and n-gram methods. In 2020 International conference on information management and technology (ICIMTech) (pp. 516-520). IEEE.

Nair, S. S. 2024. Securing Against Advanced Cyber Threats: A Comprehensive Guide to Phishing, XSS, and SQL Injection Defense. Journal of Computer Science and Technology Studies, 6(1), 76-93.

Bhardwaj, A., Chandok, S. S., Bagnawar, A., Mishra, S., & Uplaonkar, D. 2022, September. Detection of cyber attacks: XSS, sqli, phishing attacks and detecting intrusion using machine learning algorithms. In 2022 IEEE Global Conference on Computing, Power and Communication Technologies (GlobConPT) (pp. 1-6). IEEE.

Qbea'h, M., Alrabaee, S., & Mouheb, D. 2020. An analytical scanning technique to detect and prevent the transformed SQL injection and XSS attacks. In 6th International Conference on Information Systems Security and Privacy, ICISSP 2020 (pp. 603-610). SciTePress.

Alsaffar, M., Aljaloud, S., Mohammed, B. A., Al-Mekhlafi, Z. G., Almurayziq, T. S., Alshammari, G., & Alshammari, A. 2022. Detection of Web Cross-Site Scripting (XSS) Attacks. Electronics, 11(14), 2212.

Choudhary, R. R., Verma, S., & Meena, G. 2021, December. Detection of SQL injection attack using machine learning. In 2021 IEEE international conference on technology, research, and innovation for betterment of society (TRIBES) (pp. 1-6). IEEE.

Abikoye, O. C., Abubakar, A., Dokoro, A. H., Akande, O. N., & Kayode, A. A. 2020. A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm. EURASIP Journal on Information Security, 2020, 1-14.

Stiawan, D., Bardadi, A., Afifah, N., Melinda, L., Heryanto, A., Septian, T. W., ... & Budiarto, R. 2023. An Improved LSTM-PCA Ensemble Classifier for SQL Injection and XSS Attack Detection. Computer Systems Science & Engineering, 46(2).

Muslihi, M. T., & Alghazzawi, D. 2020, October. Detecting SQL injection on web application using deep learning techniques: a systematic literature review. In 2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE) (pp. 1-6). IEEE.

Muslihi, M. T., & Alghazzawi, D. 2020, October. Detecting SQL injection on web application using deep learning techniques: a systematic literature review. In 2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE) (pp. 1-6). IEEE.

Farea, A. A., Wang, C., Farea, E., & Alawi, A. B. 2021, December. Cross-site scripting (XSS) and SQL injection attacks multi-classification using bidirectional LSTM recurrent neural network. In 2021 IEEE International Conference on Progress in Informatics and Computing (PIC) (pp. 358-363). IEEE.
Schoenborn, J. M., & Althoff, K. D. 2021. Detecting SQL-Injection and Cross-Site Scripting Attacks Using Case-Based Reasoning and SEASALT. In LWDA (pp. 66-77).
Cui, Y., Cui, J., & Hu, J. 2020, February. A survey on xss attack detection and prevention in web applications. In Proceedings of the 2020 12th International Conference on Machine Learning and Computing (pp. 443-449).
Nasereddin, M., ALKhamaiseh, A., Qasaimeh, M., & Al-Qassas, R. 2023. A systematic review of detection and prevention techniques of SQL injection attacks. Information Security Journal: A Global Perspective, 32(4), 252-265.
Tadhani, J. R., Vekariya, V., Sorathiya, V., Alshathri, S., & El-Shafai, W. 2024. Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Scientific Reports, 14(1), 1803.
Rahul, S., Vajrala, C., & Thangaraju, B. 2021, November. A novel method of honeypot inclusive WAF to protect from SQL injection and XSS. In 2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON) (Vol. 1, pp. 135-140). IEEE.
Chen, D., Yan, Q., Wu, C., & Zhao, J. 2021. Sql injection attack detection and prevention techniques using deep learning. In Journal of Physics: Conference Series (Vol. 1757, No. 1, p. 012055). IOP Publishing.
Arock, M. 2021, February. Efficient detection of SQL injection attack (SQLIA) Using pattern-based neural network model. In 2021 International conference on computing, communication, and intelligent systems (ICCCIS) (pp. 343-347). IEEE.
Sivasangari, A., Jyotsna, J., & Pravalika, K. 2021, June. SQL injection attack detection using machine learning algorithm. In 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 1166-1169). IEEE.
Tanakas, P., Ilias, A., & Polemi, N. 2021, December. A novel system for detecting and preventing SQL injection and cross-site-script. In 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET) (pp. 1-6). IEEE.
Yadav, A. K., & Kumar, A. 2022. String matching algorithm based filter for preventing SQL injection and XSS attacks. In Inventive Computation and Information Technologies: Proceedings of ICICIT 2021 (pp. 793-807). Singapore: Springer Nature Singapore.
Marashdeh, Z., Suwais, K., & Alia, M. (2021, July). A survey on sql injection attack: Detection and challenges. In 2021 International Conference on Information Technology (ICIT) (pp. 957-962). IEEE.