A project report on

Detection of Cross site scripting using CNN and transformers

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning

by

AVANTHIKA RAJESH (20BAI1144)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April,2024

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled "Detection of Cross Site scripting using CNN and transformers" submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Professor Prasad.M.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: Signature of the Candidate

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **"Detection of cross site scripting using CNN and transformers" is** prepared and submitted by **Avanthika Rajesh** (**20BAI1144**) to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name: Dr./Prof. Prasad M

Date:

Signature of the Internal Examiner	Signature of the External Examiner
Name:	Name:
Date:	Date:

Approved by the Head of Department,

B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning

Name: Dr. Sweetlin Hemalatha C

Date:

(Seal of SCOPE)

ABSTRACT

Cross-site scripting (XSS) is a prevalent and potentially damaging security vulnerability in web applications. In response to this threat, researchers have proposed a novel approach utilizing Convolutional Neural Networks (CNN) and Transformers for XSS detection. The use of CNN and Transformers allows for effective modeling of the complex interactions and patterns present in web application data, facilitating the identification of XSS attacks. This approach leverages the capabilities of CNN to extract hierarchical features from the input data and the attention mechanisms of Transformers to capture long-range dependencies, thereby enhancing the detection accuracy and efficiency. By combining the strengths of both CNN and Transformers, this method demonstrates promising results in detecting XSS attacks in web applications. The collaborative nature of CNN and Transformers in this context enables the detection system to learn and adapt to evolving XSS attack patterns, thus offering an innovative and robust solution for improving web application security. Overall, the integration of CNN and Transformers in XSS detection presents a powerful and effective approach to safeguarding web applications against this common cyber threat.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Prasad M, Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Cyber Security, Mobile Communication ,AI & ML.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor i/c & Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Sweetlin Hemalatha C, Head of the Department, B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: **Avanthika Rajesh**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

1. CNN (Convolutional Neural Network)
2. XSS (Cross-Site Scripting)
3. NLP (Natural Language Processing)
4. HTML (Hypertext Markup Language)
5. CSS (Cascading Style Sheets)
6. JS (JavaScript)
7. CNNs (Convolutional Neural Networks)
8. LSTM (Long Short-Term Memory)
9. GPU (Graphics Processing Unit)
10. API (Application Programming Interface)
11. GUI (Graphical User Interface)
12. XML (eXtensible Markup Language)
13. RPC (Remote Procedure Call)
14. JSON (JavaScript Object Notation)
15. SQL (Structured Query Language)

CHAPTER 1
INTRODUCTION

1.1 Introduction

Cross-Site Scripting (XSS) represents one of the most prevalent and pernicious vulnerabilities plaguing modern web applications. It enables attackers to inject malicious scripts into web pages viewed by other users, thereby bypassing traditional security mechanisms and potentially compromising sensitive user data. Addressing XSS threats demands innovative approaches that can effectively detect and mitigate such vulnerabilities. In recent years, the integration of Convolutional Neural Networks (CNNs) and Transformers has emerged as a promising solution for enhancing the detection capabilities against XSS attacks.

CNNs have garnered significant attention in the field of cybersecurity due to their ability to effectively capture spatial dependencies within data. In the context of XSS detection, CNNs excel in identifying patterns indicative of malicious script injections across web pages. By leveraging convolutional layers, CNNs can extract relevant features from the input data, such as HTML and JavaScript code snippets, enabling them to discern between benign and malicious content. The hierarchical structure of CNNs enables them to detect subtle variations in code syntax and semantics, thereby enhancing the accuracy of XSS detection algorithms.

1.2 Overview to Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a common web application security vulnerability that occurs when an attacker injects malicious scripts into web pages viewed by other users. These scripts can then be executed in the victims' browsers, leading to various attacks such as stealing sensitive information, defacing websites, and redirecting users to malicious sites. To detect XSS attacks, researchers have explored the use of deep learning techniques such as Convolutional Neural Networks (CNN) and transformers. CNN can help identify patterns in the source code that are indicative of XSS attacks, while transformers can capture long-range dependencies in the code. By training these models on a large dataset of both benign and malicious scripts, they can learn to differentiate between safe and malicious code. This approach offers the potential to automate the detection of XSS attacks, providing a more efficient and accurate way to protect web applications from this common vulnerability.

1.3 Challenges present in Cross Site scripting

Cross-Site Scripting (XSS) presents a multifaceted challenge to the security of web applications, encompassing various technical, social, and economic dimensions. At its core, XSS exploits vulnerabilities in web applications to inject malicious scripts into web pages viewed by other users, thereby compromising their security and integrity. Despite significant advancements in cybersecurity, XSS remains prevalent due to a myriad of challenges inherent to its detection, prevention, and mitigation.

One of the primary challenges in combating XSS is the sheer diversity and complexity of modern web applications. With the proliferation of rich client-side technologies such as JavaScript frameworks and Single Page Applications (SPAs), web developers face an increasingly intricate landscape of code dependencies and interactions. This complexity often introduces vulnerabilities that attackers can exploit through XSS, ranging from insecure input validation to inadequate output encoding practices. Moreover, the dynamic nature of web applications poses challenges for traditional security mechanisms, as the execution context of client-side code can vary dynamically based on user inputs and interactions.

1.4 Project statement

The project aims to develop an innovative solution for enhancing cybersecurity measures by focusing on the detection and mitigation of Cross-Site Scripting (XSS) vulnerabilities in web applications. XSS represents a significant threat to the security and integrity of online platforms, enabling attackers to inject malicious scripts into web pages viewed by other users, potentially compromising sensitive data and undermining user trust. The primary objective of this project is to leverage advanced machine learning techniques, particularly Convolutional Neural Networks (CNNs) and Transformers, to develop a robust XSS detection system capable of identifying and mitigating both known and emerging attack vectors.

1.5 Objectives

The objectives of this project are centered around the development and implementation of an advanced Cross-Site Scripting (XSS) detection and mitigation system using machine learning techniques, particularly Convolutional Neural Networks (CNNs) and Transformers. The primary goals are to enhance cybersecurity measures for web applications, mitigate XSS vulnerabilities, and protect sensitive user data from malicious exploitation.

- Analyze existing XSS vulnerabilities and attack patterns through literature review and historical data to inform robust detection system development.
- Curate a diverse dataset of benign and malicious code samples for training machine learning models used in detection systems, ensuring representativeness and diversity.
- Develop machine learning models utilizing CNNs for feature extraction and Transformers for contextual analysis to improve accuracy and robustness.
- Train and optimize machine learning models with fine-tuning, algorithm optimization, and transfer learning to achieve high detection accuracy with minimal false positives and false negatives.
- Evaluate detection system performance across diverse scenarios and real-world datasets, using metrics like precision, recall, and F1 score, while testing resilience against evasion techniques.
- Integrate XSS detection system into cybersecurity workflows and web development pipelines, focusing on user-friendly interfaces, APIs, and scalability for widespread

adoption.

Overall, the objectives of this project are aimed at advancing the state-of-the-art in XSS detection and mitigation through the development of a novel machine learning-based solution. By addressing the inherent challenges of XSS vulnerabilities and adopting a holistic approach to cybersecurity, the project aims to enhance the resilience and security posture of web applications, thereby safeguarding sensitive user data and preserving trust in the digital ecosystem.

1.6 Scope of the project

The scope of this project encompasses the development and implementation of an advanced Cross-Site Scripting (XSS) detection and mitigation system leveraging machine learning techniques, specifically Convolutional Neural Networks (CNNs) and Transformers. The primary focus is on enhancing cybersecurity measures for web applications, detecting and mitigating XSS vulnerabilities, and protecting sensitive user data from malicious exploitation.

The project scope includes a thorough examination of existing XSS vulnerabilities and attack patterns to gain insights into the evolving threat landscape. This analysis will inform the development of the detection system, ensuring it can effectively identify both known and emerging XSS attack vectors. The scope encompasses research into historical attack data, literature review, and analysis of common exploit techniques used by attackers to understand the intricacies of XSS vulnerabilities thoroughly.

CHAPTER 2
LITERATURE SURVEY

1. Kaur, J., Garg, U., & Bathla, G. (2023). Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. Artificial Intelligence Review, 1-45.

- The review provides a comprehensive overview of various machine learning techniques employed for the detection of cross-site scripting (XSS) attacks in web applications.
- Different types of XSS attacks, including stored, reflected, and DOM-based XSS, are discussed along with their detection challenges and implications for web security.
- The review evaluates the effectiveness of machine learning algorithms such as Support Vector Machines (SVM), Random Forest, and Neural Networks in detecting XSS vulnerabilities.
- It highlights the importance of feature selection and preprocessing techniques in enhancing the accuracy and efficiency of XSS detection systems.
- The review identifies the limitations of existing XSS detection methods and suggests directions for future research to address emerging threats and improve detection capabilities.
- Insights are provided into the integration of machine learning with other security measures to develop robust XSS detection systems capable of mitigating evolving attack vectors.

2. Al-Haija, Q. A. (2023). Cost-effective detection system of cross-site scripting attacks using hybrid learning approach. Results in Engineering, 19, 101266.

- The study proposes a cost-effective approach for detecting cross-site scripting (XSS) attacks by combining multiple machine learning algorithms in a hybrid learning framework.

- Different types of XSS attacks and their potential impact on web security are discussed, emphasizing the need for efficient detection mechanisms to safeguard web applications.

- The hybrid learning approach integrates the strengths of different machine learning algorithms, including decision trees, neural networks, and ensemble methods, to enhance detection accuracy.

- The study evaluates the performance of the proposed detection system using real-world datasets, demonstrating its effectiveness in identifying XSS vulnerabilities while minimizing false positives.

- The cost-effectiveness of the approach is analyzed in terms of computational resources, training time, and scalability, highlighting its suitability for deployment in resource-constrained environments.

- Insights are provided into the design considerations and optimization strategies for implementing the hybrid learning approach in practical XSS detection systems.

3. Chaudhary, P., Gupta, B. B., & Singh, A. K. (2023). Adaptive cross-site scripting attack detection framework for smart devices security using intelligent filters and attack ontology. Soft Computing, 27(8), 4593-4608.

- The study proposes an adaptive framework for detecting cross-site scripting (XSS) attacks targeting smart devices, leveraging intelligent filters and attack ontology to enhance security.

- Smart devices' vulnerabilities to XSS attacks and their implications for user privacy and data integrity are discussed, underscoring the need for adaptive detection mechanisms.

- The framework employs intelligent filters to analyze web content and identify potential XSS payloads, mitigating the risk of malicious script injection.

- Attack ontology is utilized to categorize and prioritize XSS attack patterns, enabling proactive detection and response to emerging threats targeting smart devices.

- The study evaluates the performance of the adaptive detection framework using real-world datasets, demonstrating its effectiveness in mitigating XSS vulnerabilities in diverse smart device environments.

- Insights are provided into the integration of adaptive detection mechanisms with existing security protocols to strengthen the resilience of smart devices against XSS attacks and other cyber threats.

4. Alaoui, R. L. (2023). Cross Site Scripting Attack Detection Approach Based on LSTM Encoder-Decoder and Word Embeddings. International Journal of Intelligent Systems and Applications in Engineering, 11(2), 277-282.

- LSTM Encoder-Decoder Architecture: The study proposes an innovative approach for detecting cross-site scripting (XSS) attacks using a Long Short-Term Memory (LSTM) encoder-decoder architecture. This architecture enables the model to effectively capture sequential patterns in web content, facilitating accurate XSS detection.

- Word Embeddings: It utilizes word embeddings to represent web content as dense vectors, allowing the LSTM model to learn meaningful representations of words and phrases in XSS attack payloads and benign web pages. This enhances the model's ability to distinguish between malicious and legitimate content.

- Sequential Feature Learning: The approach leverages the sequential nature of LSTM networks to learn temporal dependencies and contextual information from input sequences. This enables the model to identify subtle patterns indicative of XSS attacks, improving detection accuracy.

- Experimental Validation: The study conducts experiments to evaluate the performance of the proposed approach using benchmark datasets. It demonstrates the efficacy of the LSTM encoder-decoder architecture and word embeddings in accurately detecting XSS attacks while minimizing false positives.

- Practical Implications: By integrating LSTM-based sequential modeling and word embeddings, the approach offers practical implications for enhancing XSS detection in real-world web applications. It provides a robust defense mechanism against XSS attacks, thereby improving overall web security.

- Future Directions: The research identifies potential avenues for future research, such as exploring ensemble methods, attention mechanisms, and adversarial training techniques to further enhance the performance and robustness of XSS detection systems based on LSTM encoder-decoder architectures.

5. Thajeel, I. K., Samsudin, K., Hashim, S. J., & Hashim, F. (2023). Dynamic feature selection model for adaptive cross-site scripting attack detection using developed multi-agent deep Q learning model. Journal of King Saud University-Computer and Information Sciences, 35(6), 101490.

- Multi-Agent Deep Q Learning Model: The study presents a dynamic feature selection model for adaptive cross-site scripting (XSS) attack detection, leveraging a multi-agent deep Q learning framework. This model enables intelligent feature selection and classification, enhancing detection accuracy.

- Dynamic Feature Selection: The proposed model dynamically selects relevant features for XSS attack detection, adapting to changing attack patterns and web content characteristics. This ensures that the detection system focuses on the most informative features, improving overall effectiveness.

- Deep Q Learning Framework: It utilizes deep Q learning techniques to train multiple agents to select features that maximize detection accuracy while minimizing false positives. This framework enables agents to learn optimal feature selection policies through trial and error.

- Adaptive Learning Mechanism: The model incorporates an adaptive learning mechanism that allows it to continuously update feature selection and classification policies based on feedback from the environment. This ensures robust and adaptive detection capabilities in dynamic threat landscapes.

- Experimental Validation: The study conducts experiments to validate the performance of the proposed model using benchmark datasets. It demonstrates the effectiveness of dynamic feature selection and multi-agent deep Q learning in accurately detecting XSS attacks while minimizing false alarms.

- Practical Implications: By integrating dynamic feature selection and multi-agent deep Q learning, the model offers practical implications for enhancing XSS detection in real-world scenarios. It provides a scalable and adaptive solution for mitigating XSS vulnerabilities in web applications.

6. Pardomuan, C. R., Kurniawan, A., Darus, M. Y., Mohd Ariffin, M. A., & Muliono, Y. (2023). Server-Side Cross-Site Scripting Detection Powered by HTML Semantic Parsing Inspired by XSS Auditor. Pertanika Journal of Science & Technology, 31(3).

- HTML Semantic Parsing: The study introduces a server-side cross-site scripting (XSS) detection approach powered by HTML semantic parsing inspired by XSS Auditor. This technique involves analyzing the syntactic structure and semantics of HTML documents to identify XSS vulnerabilities.

- XSS Auditor Inspiration: Drawing inspiration from the XSS Auditor feature in modern web browsers, the approach leverages similar principles for server-side XSS detection. By parsing and analyzing HTML documents, the system aims to identify potential XSS attack vectors and mitigate associated risks.

- Semantic Analysis Techniques: It utilizes semantic analysis techniques to extract meaningful information about HTML elements and attributes, enabling the detection system to identify suspicious patterns indicative of XSS vulnerabilities. This enhances the accuracy and effectiveness of XSS detection.

- Server-Side Implementation: Unlike traditional client-side XSS detection mechanisms, the proposed approach operates at the server-side, allowing for centralized detection and mitigation of XSS vulnerabilities across web applications hosted on the server. This ensures comprehensive protection against XSS attacks.

- Experimental Validation: The study validates the performance of the server-side XSS detection approach using real-world datasets and experimental scenarios. It demonstrates the effectiveness of HTML semantic parsing in accurately detecting XSS vulnerabilities while minimizing false positives.

- Practical Implications: By offering a server-side XSS detection mechanism powered by HTML semantic parsing, the approach provides practical implications for enhancing web application security. It offers a proactive defense mechanism against XSS attacks, complementing existing client-side security measures.

7.Hu, T., Xu, C., Zhang, S., Tao, S., & Li, L. (2023). Cross-site scripting detection with two-channel feature fusion embedded in self-attention mechanism. Computers & Security, 124, 102990.

- Two-Channel Feature Fusion: The study proposes a cross-site scripting (XSS) detection approach that incorporates a two-channel feature fusion strategy. This strategy combines both content-based features and context-based features extracted from web content, enhancing the model's detection capabilities.

- Self-Attention Mechanism: The approach embeds a self-attention mechanism into the XSS detection model, allowing the model to focus on different parts of the input sequence. This mechanism enables the model to capture relevant features effectively, improving detection accuracy.

- Feature Representation Learning: By leveraging the self-attention mechanism, the model autonomously learns the importance of different features within the input sequence. This adaptive learning approach enhances the model's ability to discern between benign and malicious content.

- Experimental Validation: The study conducts experiments to validate the performance of the proposed approach using benchmark datasets. It demonstrates the efficacy of the two-channel feature fusion embedded in the self-attention mechanism in accurately detecting XSS attacks while minimizing false positives.

- Practical Deployment Considerations: The research discusses practical considerations for deploying the proposed XSS detection approach in real-world environments. This includes computational efficiency, scalability, and integration with existing security infrastructure, ensuring feasibility and effectiveness.

- Future Directions: Based on the findings, the study identifies potential areas for future research, such as exploring advanced feature fusion techniques and incorporating domain-specific knowledge into the detection model. These directions aim to further improve the robustness and adaptability of XSS detection systems.

8.Kumar, A., & Sharma, I. (2023, April). Performance Evaluation of Machine Learning Techniques for Detecting Cross-Site Scripting Attacks. In 2023 11th International Conference on Emerging Trends in Engineering & Technology-Signal and Information Processing (ICETET-SIP) (pp. 1-5). IEEE.

- Machine Learning Techniques: The study evaluates various machine learning techniques for detecting cross-site scripting (XSS) attacks, including Support Vector Machines (SVM), Decision Trees, Random Forests, and Neural Networks. It assesses the performance of each technique in terms of detection accuracy and false positive rate.

- Experimental Setup: The research describes the experimental setup used to evaluate the performance of machine learning techniques, including dataset selection, feature extraction methods, and evaluation metrics. This ensures consistency and reproducibility of the experimental results.

- Comparative Analysis: It conducts a comparative analysis of different machine learning techniques, highlighting their strengths and weaknesses in detecting XSS attacks. The analysis aids in identifying the most effective approach for XSS detection based on specific application requirements.

- Impact of Feature Engineering: The study investigates the impact of feature engineering techniques on the performance of machine learning models for XSS detection. It explores different feature sets and feature selection methods to optimize detection accuracy.

- Real-World Application Scenarios: The research discusses the practical implications of the performance evaluation results for real-world application scenarios. It provides insights into selecting suitable machine learning techniques and feature engineering strategies for deploying effective XSS detection systems.

- Future Research Directions: Based on the findings of the performance evaluation, the study identifies potential areas for future research and improvement in XSS detection techniques. This includes exploring ensemble learning methods, deep learning architectures, and hybrid approaches for enhanced detection performance.

9.Panwar, P., Mishra, H., & Patidar, R. (2023). An Analysis of the Prevention and Detection of Cross Site Scripting Attack. International Journal, 11(1).

- Prevention Strategies Analysis: The study conducts a detailed analysis of prevention strategies employed to mitigate cross-site scripting (XSS) attacks. It examines various client-side and server-side techniques, such as input validation, output encoding, and secure coding practices.

- Detection Mechanisms Evaluation: It evaluates different detection mechanisms utilized to identify XSS vulnerabilities in web applications. The analysis covers signature-based detection, anomaly detection, and machine learning-based approaches, assessing their effectiveness and practicality.

- Comparative Study: The research compares the strengths and weaknesses of prevention and detection strategies, highlighting trade-offs between security effectiveness, implementation complexity, and performance overhead. This comparison aids in selecting appropriate strategies based on specific application requirements.

- Impact on Web Application Security: It discusses the implications of XSS prevention and detection measures on overall web application security posture. This includes considerations for balancing security measures with usability, performance, and user experience considerations.

- Case Studies and Best Practices: The study presents case studies illustrating real-world examples of XSS prevention and detection strategies implemented in web applications. It identifies best practices and lessons learned from these case studies to inform security practitioners and developers.

- Future Directions: Based on the analysis, the research identifies emerging trends and future directions in XSS prevention and detection research. This includes exploring novel techniques such as client-side security mechanisms, machine learning-driven detection, and automated vulnerability remediation.

10.Lu, D., & Liu, L. (2023, February). Research on Cross-site Scripting Attack Detection Technology Based on Few-shot Learning. In 2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 6, pp. 1425-1429). IEEE.

- Few-shot Learning Approach: The study investigates the application of few-shot learning techniques for cross-site scripting (XSS) attack detection. Few-shot learning enables the model to generalize from a limited number of training examples, making it suitable for detecting novel XSS attack patterns.

- Dataset Preparation: It describes the preparation of datasets for few-shot learning, including the selection of representative XSS attack instances and benign web pages. The study also explores techniques for data augmentation to enhance the diversity and generalization capability of the model.

- Model Architecture Design: The research proposes a novel model architecture tailored for few-shot learning-based XSS detection. It incorporates meta-learning mechanisms and attention mechanisms to enable the model to effectively learn from few examples and adapt to new XSS attack scenarios.

- Evaluation Methodology: The study outlines the evaluation methodology used to assess the performance of the few-shot learning approach in detecting XSS attacks. This includes benchmarking against traditional machine learning techniques and evaluating the model's robustness to different attack variations.

- Experimental Results: It presents experimental results demonstrating the effectiveness of the proposed few-shot learning approach in detecting XSS attacks. The results showcase the model's ability to generalize from limited training data and achieve competitive performance compared to traditional methods.

- Practical Implications: The research discusses the practical implications of adopting few-shot learning for XSS attack detection, including its potential to enhance the adaptability and scalability of detection systems in real-world scenarios with limited labeled data availability.

11.Lee, S., Wi, S., & Son, S. (2022, April). Link: Black-box detection of cross-site scripting vulnerabilities using reinforcement learning. In Proceedings of the ACM Web Conference 2022 (pp. 743-754).

- Black-box Detection Approach: The study proposes a black-box detection method for identifying cross-site scripting (XSS) vulnerabilities in web applications using reinforcement learning techniques. This approach does not rely on access to the application's internal code or structure.

- Reinforcement Learning Framework: It outlines the reinforcement learning framework used for training an agent to interact with the web application and identify potential XSS vulnerabilities. The agent learns optimal actions through trial and error to maximize its cumulative reward.

- Environment Simulation: The research describes the simulation environment used to emulate interactions with the web application, including crawling and exploring different input patterns. This enables the reinforcement learning agent to learn effective strategies for XSS vulnerability detection.

- Reward Function Design: The study discusses the design of the reward function used to provide feedback to the reinforcement learning agent. The reward function incentivizes actions that lead to the discovery of XSS vulnerabilities while penalizing ineffective or harmful actions.

- Experimental Validation: It presents experimental results demonstrating the effectiveness of the black-box detection approach in identifying XSS vulnerabilities across different web applications. The results showcase the model's ability to achieve high detection rates while minimizing false positives.

- Practical Deployment Considerations: The research discusses practical considerations for deploying the black-box detection approach in real-world scenarios, including integration with existing security testing frameworks, scalability, and adaptability to diverse web application architectures.

12.Chen, H. C., Nshimiyimana, A., Damarjati, C., & Chang, P. H. (2021, January). Detection and prevention of cross-site scripting attack with combined approaches. In 2021 International Conference on Electronics, Information, and Communication (ICEIC) (pp. 1-4). IEEE.

- Combined Approaches: The research investigates the effectiveness of combining multiple approaches for both detecting and preventing cross-site scripting (XSS) attacks. It integrates signature-based detection, anomaly detection, input validation, and output encoding techniques to provide comprehensive protection against XSS vulnerabilities.

- Detection Mechanisms: The study evaluates different detection mechanisms, including signature-based detection for identifying known XSS attack patterns and anomaly detection for detecting deviations from normal web traffic behavior. The combination of these mechanisms enhances the detection accuracy and robustness of the overall system.

- Prevention Strategies: It explores various prevention strategies such as input validation and output encoding to mitigate XSS vulnerabilities at different stages of web application development and execution. By implementing multiple layers of defense, the system aims to minimize the likelihood of successful XSS attacks.

- Experimental Validation: The research conducts experiments to validate the effectiveness of the combined approaches in detecting and preventing XSS attacks. It evaluates the system's performance in terms of detection rate, false positive rate, and prevention effectiveness using real-world datasets and attack scenarios.

- Integration Challenges: The study addresses challenges related to integrating multiple detection and prevention mechanisms into a cohesive security framework. It discusses issues such as compatibility, performance overhead, and management complexity, offering insights into overcoming these challenges for practical deployment.

- Practical Implications: Based on the experimental results and analysis, the research provides practical implications for deploying combined approaches for XSS detection and prevention in real-world web applications. It emphasizes the importance of holistic security measures to mitigate XSS risks effectively.

13.Rodríguez, G. E., Torres, J. G., Flores, P., & Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. Computer Networks, 166, 106960.

- Survey Scope: The paper presents a comprehensive survey on cross-site scripting (XSS) attacks and mitigation techniques. It covers various aspects of XSS attacks, including attack vectors, impact, common exploitation methods, and existing mitigation strategies.

- Attack Vectors Analysis: The survey provides an in-depth analysis of different XSS attack vectors, such as reflected XSS, stored XSS, and DOM-based XSS, highlighting their characteristics, prevalence, and potential impact on web applications and users.

- Exploitation Methods: It explores common exploitation methods employed by attackers to exploit XSS vulnerabilities, including payload crafting techniques, evasion strategies, and obfuscation methods. Understanding these methods is crucial for developing effective mitigation strategies.

- Mitigation Techniques Overview: The research discusses a wide range of mitigation techniques and best practices for preventing XSS attacks, including input validation, output encoding, Content Security Policy (CSP), and web application firewalls (WAFs). It evaluates the effectiveness of each technique and their applicability in different contexts.
- Emerging Trends: The survey identifies emerging trends and advancements in XSS attack techniques and mitigation strategies, such as the adoption of modern security headers, client-side security mechanisms, and automated vulnerability scanning tools. These trends inform future research and development efforts in XSS defense.

- Practical Recommendations: Based on the survey findings, the paper provides practical recommendations for developers, security practitioners, and organizations to enhance their defense against XSS attacks. This includes adopting a defense-in-depth approach, implementing secure coding practices, and staying informed about evolving attack trends.

14.Gan, J. M., Ling, H. Y., & Leau, Y. B. (2021). A Review on detection of cross-site scripting attacks (XSS) in web security. In Advances in Cyber Security: Second International Conference, ACeS 2020, Penang, Malaysia, December 8-9, 2020, Revised Selected Papers 2 (pp. 685-709). Springer Singapore.

- Review Scope: The paper provides a comprehensive review of the detection techniques employed to identify cross-site scripting (XSS) attacks in web security. It covers a wide range of approaches, including signature-based detection, anomaly detection, machine learning, and hybrid methods.

- Comparative Analysis: The review conducts a comparative analysis of different XSS detection techniques, assessing their strengths, weaknesses, and performance characteristics. It evaluates factors such as detection accuracy, false positive rate, computational overhead, and scalability.

- Signature-based Detection: It discusses signature-based detection methods that rely on predefined patterns or rules to identify known XSS attack payloads. The review examines the effectiveness of signature-based approaches in detecting common XSS attack vectors and their limitations in detecting novel attacks.

- Anomaly Detection: The paper explores anomaly detection techniques that detect deviations from normal web application behavior, which may indicate the presence of XSS attacks. It evaluates the applicability of anomaly detection in identifying previously unseen attack patterns and its resilience to evasion techniques.

- Machine Learning Approaches: The review delves into machine learning-based XSS detection methods, which leverage algorithms such as Support Vector Machines (SVM), Decision Trees, Random Forests, and Neural Networks to learn patterns indicative of XSS attacks. It assesses the performance of these techniques and their suitability for real-world deployment.

- Hybrid Methods: It discusses hybrid approaches that combine multiple detection techniques to leverage their respective strengths and mitigate their weaknesses. The review explores synergies between signature-based, anomaly detection, and machine learning methods to enhance overall detection effectiveness.

15.Usha, G., Kannimuthu, S., Mahendiran, P. D., Shanker, A. K., & Venugopal, D. (2020). Static analysis method for detecting cross site scripting vulnerabilities. International Journal of Information and Computer Security, 13(1), 32-47.

- Static Analysis Method: The paper introduces a static analysis method for detecting cross-site scripting (XSS) vulnerabilities in web applications. This method involves analyzing the source code of web applications without executing them, aiming to identify potential XSS attack vectors.

- Source Code Analysis Techniques: It discusses various source code analysis techniques employed in static analysis for XSS vulnerability detection, including lexical analysis, syntax analysis, and semantic analysis. These techniques enable the identification of insecure coding practices and potential XSS vulnerabilities.

- Vulnerability Identification: The research outlines the process of vulnerability identification through static analysis, including pattern matching, data flow analysis, and taint tracking. By tracing the flow of user input through the application code, potential XSS vulnerabilities can be identified and mitigated.

- Code Sanitization Recommendations: It provides recommendations for code sanitization techniques to mitigate XSS vulnerabilities identified through static analysis. This includes input validation, output encoding, and context-aware sanitization to prevent malicious input from being interpreted as code.

- Automation and Tool Support: The paper discusses the automation of static analysis techniques for XSS vulnerability detection and the availability of tools to assist developers in identifying and remediating vulnerabilities. Automated static analysis tools can help streamline the vulnerability management process and improve the overall security posture of web applications.

- Practical Implications: By offering a static analysis method for XSS vulnerability detection, the research provides practical implications for improving the security of web applications. It emphasizes the importance of secure coding practices and proactive vulnerability management to mitigate XSS risks effectively.

16.Sahoo, B., & Pattnaik, P. K. (2023). Cybersecurity Analysis and Phishing Attack. In Communication Technology and Gender Violence (pp. 31-37). Cham: Springer International Publishing.

- Examines cybersecurity analysis with a focus on phishing attacks.
- Discusses the prevalence and impact of phishing attacks in the digital landscape.
- Explores different types of phishing techniques and their characteristics.
- Highlights the role of communication technology in mitigating gender-based cyber violence.
- Proposes strategies for improving cybersecurity measures against phishing attacks.

17.Abd Alkareem, M. H. B., Nasif, F. Q., Ahmed, S. R., Miran, L. D., Algburi, S., & ALmashhadany, M. T. (2023, November). Linguistics for Crimes in the World by AI-Based Cyber Security. In 2023 7th International Symposium on Innovative Approaches in Smart Technologies (ISAS) (pp. 1-5). IEEE.

- Investigates the role of linguistics in AI-based cybersecurity for combating global crimes.
- Explores the potential of AI in analyzing linguistic patterns associated with cybercrimes.
- Discusses the integration of linguistic analysis techniques with cybersecurity frameworks.
- Examines the challenges and ethical considerations in employing AI for crime detection.
- Proposes future research directions for advancing linguistics-based cybersecurity solutions.

18.Baniya, D., & Chaudhary, A. (2023). Detecting cross-site scripting attacks using machine learning: A systematic review. Artificial Intelligence, Blockchain, Computing and Security Volume 1, 743-748.

- Conducts a systematic review on detecting cross-site scripting attacks using machine learning.
- Summarizes existing machine learning techniques for XSS detection.
- Evaluates the performance of machine learning models in identifying XSS vulnerabilities.
- Discusses the limitations and potential biases in current XSS detection approaches.
- Proposes recommendations for enhancing the accuracy and robustness of ML-based XSS detection systems.

19.Kirchner, R., Möller, J., Musch, M., Klein, D., Rieck, K., & Johns, M. Dancer in the Dark: Synthesizing and Evaluating Polyglots for Blind Cross-Site Scripting.

- Presents a study on synthesizing and evaluating polyglots for blind cross-site scripting.
- Investigates the effectiveness of polyglot techniques in bypassing XSS filters.
- Discusses the implications of blind XSS attacks for web security.
- Analyzes the challenges associated with detecting and mitigating blind XSS vulnerabilities.
- Proposes strategies for improving the resilience of web applications against blind XSS attacks.

20.Khare, S., & Badholia, A. (2023). BLA2C2: Design of a novel blockchain-based light-weight authentication & access control layer for cloud deployments. International Journal on Recent and Innovation Trends in Computing and Communication, 11(3), 283-294.

- Introduces BLA2C2, a novel blockchain-based lightweight authentication and access control layer for cloud deployments.
- Discusses the design principles and architecture of BLA2C2.
- Evaluates the performance and scalability of BLA2C2 in cloud environments.
- Explores the potential benefits of integrating blockchain technology with access control mechanisms.
- Proposes future research directions for enhancing the security and efficiency of cloud deployments using BLA2C2.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing system for the detection of cross-site scripting (XSS) commonly involves the utilization of machine learning techniques such as convolutional neural networks (CNN) and transformers for enhanced accuracy and efficiency. CNNs are effective in extracting features from the input data, which in the case of XSS detection, involves analyzing the code to identify potentially malicious scripts. Transformers, on the other hand, are capable of capturing long-range dependencies within the code, making them suitable for detecting complex patterns and sequences indicative of XSS attacks. By combining the strengths of CNNs and transformers, the existing system can effectively analyze web content in real-time to identify and mitigate XSS vulnerabilities. The system typically involves preprocessing the input data, encoding it into suitable formats for the CNN and transformer models, training the models on large datasets of both benign and malicious code samples, and finally deploying the trained models for real-time detection of XSS attacks. The use of machine learning models such as CNNs and transformers in XSS detection has shown promising results in terms of accuracy and speed, enabling organizations to enhance their cybersecurity posture and protect their web applications from malicious attacks. However, continuous research and development efforts are needed to keep pace with evolving XSS attack techniques and ensure the effectiveness of the detection system in detecting new and sophisticated threats.The existing system for the detection of cross-site scripting (XSS) poses several disadvantages that hinder its effectiveness. One key limitation is the reliance on traditional machine learning algorithms, which may lack the ability to effectively capture complex patterns and variations inherent in XSS attacks. These algorithms typically struggle to handle the dynamic and evolving nature of XSS attacks, leading to a high rate of false positives and false negatives. Additionally, the feature engineering process required by these algorithms can be time-consuming and labor-intensive, often resulting in suboptimal performance. Another disadvantage is the limited scalability of these systems, making it challenging to handle the exponentially increasing volume and complexity of web applications and attacks. Furthermore, the lack of interpretability in existing systems can make it difficult to understand the reasoning behind detection decisions, impeding the ability to fine-tune and improve the system. In contrast, leveraging Convolutional Neural Networks (CNNs) and transformers for XSS detection offers promising advantages, such as the ability to automatically extract intricate features from input data and capture complex relationships within the data. By utilizing deep learning models, the system can potentially enhance detection accuracy, reduce false positives, and adapt better to evolving attack techniques. However, challenges related to model interpretability, computational resources, and data labeling still need to be addressed to fully leverage the capabilities of CNNs and transformers for XSS detection.

3.2 PROPOSED SYSTEM

The proposed system utilizes a sophisticated approach for detecting cross-site scripting (XSS) attacks by combining Convolutional Neural Networks (CNN) and transformers. The architecture of the system involves using CNN for feature extraction and transformers for sequence modeling, enabling the model to effectively capture both spatial and sequential patterns indicative of XSS attacks. By leveraging CNN's ability to learn hierarchical representations of input data and transformers' proficiency in processing sequential information, the system can efficiently identify and classify XSS vulnerabilities in web applications. The CNN component is responsible for extracting relevant features from the input data, such as code snippets and script patterns, while the transformer component processes the extracted features to understand the contextual relationships within the code snippets. This hybrid approach enables the model to not only detect known XSS patterns but also adapt to new and evolving attack techniques. Furthermore, the system is designed to continuously learn from new data and update its detection capabilities through a feedback loop mechanism, ensuring its effectiveness in detecting emerging XSS threats. Overall, by combining the strengths of CNN and transformers, the proposed system offers a robust and adaptive solution for the proactive detection of XSS vulnerabilities in web applications, enhancing the security posture of organizations and reducing the risk of cyberattacks.The proposed system for the detection of cross-site scripting (XSS) using Convolutional Neural Networks (CNN) and transformers offers a range of significant advantages. Firstly, the utilization of CNNs allows for the extraction of complex features from the input data, enabling the system to effectively identify patterns and anomalies associated with XSS attacks. By leveraging CNN's ability to learn hierarchical representations, the system can enhance its detection capabilities and accurately pinpoint malicious scripts embedded within web applications. Moreover, the use of transformers enhances the system's performance by enabling it to capture long-range dependencies in the input data, thereby improving the overall detection accuracy and reducing false positives. Additionally, the combination of CNNs and transformers as part of a single system provides a comprehensive approach to XSS detection, leveraging the strengths of both architectures to achieve a more robust and reliable detection mechanism. This hybrid approach enhances the system's adaptability to different types of XSS attacks and ensures a high level of precision in identifying malicious scripts, thereby enhancing the overall security posture of web applications. Furthermore, the proposed system's ability to continuously learn and adapt to new and evolving threats makes it a valuable tool for cybersecurity professionals seeking to proactively defend against XSS attacks. Overall, the integration of CNN and transformers in the detection of XSS represents a significant advancement in the field of web security, offering improved efficiency, accuracy, and scalability in identifying and mitigating potential vulnerabilities.

3.3 FEASIBILITY STUDY

A feasibility study for the detection of cross-site scripting using convolutional neural networks (CNN) and transformers involves assessing the practicality and viability of implementing these technologies for this specific purpose. CNNs are effective in image processing tasks due to their ability to capture spatial hierarchies, while transformers excel in handling sequential data and have been successful in natural language processing applications. The feasibility study would explore the adaptability of these architectures to identifying and mitigating cross-site scripting attacks in web applications. Factors to consider include the availability of labeled datasets for training the models, computational resources required for training and inference, as well as the performance metrics such as precision, recall, and F1-score. Additionally, the study would investigate the scalability of the proposed solution, potential challenges in model integration with existing web security systems, and the overall impact on detection accuracy and false positive rates. Stakeholder input, cost analysis, and a risk assessment would also be crucial components of the feasibility study to determine the long-term sustainability and practicality of implementing CNNs and transformers for cross-site scripting detection. Ultimately, the study aims to provide insights into the technical capabilities, economic viability, and operational feasibility of leveraging deep learning techniques for enhancing web security against cross-site scripting vulnerabilities.

3.3.1 ECONOMICAL FEASIBILITY

Economical feasibility is a crucial aspect to consider when determining the viability of implementing a new technology or method, such as using Convolutional Neural Networks (CNN) and transformers for the detection of cross-site scripting (XSS) vulnerabilities. In the field of cybersecurity, ensuring that the benefits of using advanced technologies outweigh the costs is essential for organizations looking to enhance their security measures effectively.

The integration of CNN and transformers for XSS detection can bring several potential advantages, such as improved accuracy in identifying malicious scripts, faster detection speeds, and the ability to adapt to new and evolving XSS attack techniques. By leveraging the capabilities of these advanced technologies, organizations can enhance their overall security posture and better protect their systems and data from XSS vulnerabilities.

However, it is essential to consider the economic implications of implementing such a solution. Factors such as the initial investment required for acquiring and setting up the necessary hardware and software infrastructure, as well as ongoing maintenance and training costs, need to be carefully evaluated. Furthermore, organizations should assess the potential return on investment (ROI) in terms of reduced security incidents, minimized business disruptions, and enhanced customer trust.

In conducting an economic feasibility analysis, organizations should weigh the costs of implementing CNN and transformers for XSS detection against the expected benefits and savings. This assessment should take into account not only the direct financial costs but also the potential value added in terms of improved security posture, regulatory compliance, and brand reputation.

Ultimately, the decision to adopt CNN and transformers for XSS detection should be based on a comprehensive analysis of the economic feasibility, taking into consideration both the short-term costs and long-term benefits. By carefully evaluating the economic implications, organizations can make informed decisions that align with their strategic objectives and cybersecurity priorities.

3.3.2 TECHNICAL FEASIBILITY

Technical feasibility refers to the assessment of the practicality and viability of implementing a particular technology or solution. In the context of detecting cross-site scripting (XSS) using Convolutional Neural Networks (CNN) and transformers, it is important to evaluate the technical feasibility of this approach.

CNNs and transformers are advanced machine learning models that have shown great potential in various domains, including computer vision and natural language processing. Applying these models to the task of detecting XSS attacks can offer several advantages, such as improved accuracy, scalability, and efficiency.

One key aspect of technical feasibility is the availability of data. In the case of detecting

XSS attacks, a robust dataset containing both benign and malicious web requests is essential for training and testing the CNN and transformer models. Access to such data is crucial for ensuring the effectiveness of the detection system.

Another important factor to consider is the computational resources required to train and deploy the models. CNNs and transformers are computationally intensive models, especially when dealing with large-scale datasets. Adequate hardware resources, such as GPUs and cloud computing services, may be necessary to effectively train and run the models for XSS detection.

Furthermore, the integration of CNNs and transformers into existing web application security systems must be carefully considered. This involves developing a streamlined workflow for processing incoming web requests, feeding the data into the models, and interpreting the output for decision-making.

Lastly, ongoing maintenance and updates to the detection system must be taken into account. As new XSS attack vectors emerge and the threat landscape evolves, the models must be continuously retrained and improved to adapt to these changes.

In conclusion, while the use of CNNs and transformers for detecting XSS attacks offers promising benefits, ensuring the technical feasibility of this approach requires close attention to data availability, computational resources, system integration, and maintenance procedures. By addressing these key considerations, developers can build a robust and effective XSS detection system using advanced machine learning techniques.

3.3.3 OPERATIONAL FEASIBILITY

Operational feasibility is a crucial aspect to consider when implementing advanced technologies like Convolutional Neural Networks (CNN) and transformers for the detection of cross-site scripting (XSS) vulnerabilities. In the context of this project, operational feasibility refers to the practicality of implementing and utilizing these deep learning models in real-world scenarios to effectively detect and mitigate XSS threats.

When considering operational feasibility for the detection of XSS using CNN and transformers, several key factors need to be assessed:

1. Resource Requirements: Implementing and training CNN and transformer models require significant computational resources, including high-performance GPUs and large datasets. It is essential to evaluate whether the organization has the necessary resources to support the deployment and maintenance of these models.

2. Technical Expertise: Operating CNN and transformer models requires specialized knowledge and expertise in deep learning, neural networks, and natural language processing. Organizations must ensure that they have access to skilled professionals who can effectively develop, train, and fine-tune these models for XSS detection.

3. Integration with Existing Systems: It is essential to assess how easily CNN and transformer models can be integrated with existing security infrastructure and tools within the organization. Compatibility issues, data integration challenges, and interoperability with other systems need to be carefully considered.

4. Scalability and Performance: The operational feasibility of utilizing CNN and transformer models for XSS detection also depends on their scalability and performance. Organizations must evaluate whether these models can efficiently handle increasing volumes of web traffic and data while maintaining high levels of accuracy and speed.

5. Cost Considerations: Implementing advanced deep learning models like CNN and transformers can involve significant costs related to infrastructure, training, and

maintenance. A cost-benefit analysis should be conducted to determine whether the potential benefits of utilizing these models for XSS detection outweigh the associated costs.

In conclusion, operational feasibility plays a critical role in determining the success of using CNN and transformers for the detection of cross-site scripting vulnerabilities. By carefully assessing resource requirements, technical expertise, system integration, scalability, performance, and cost considerations, organizations can make informed decisions about the practicality and implementation of these advanced technologies for XSS detection.

3.3.4 SOCIAL FEASIBILITY

Social feasibility refers to the acceptance and support of a technology or methodology by the society or community in which it is intended to be implemented. In the context of the detection of cross-site scripting (XSS) using convolutional neural networks (CNN) and transformers, social feasibility is an important consideration that goes beyond the technical capabilities of the proposed solution.

Implementing a system based on CNN and transformers for the detection of XSS presents both opportunities and challenges in terms of social feasibility. On one hand, utilizing advanced technologies like CNN and transformers can significantly enhance the accuracy and efficiency of XSS detection, which can lead to a safer and more secure online environment for users. This can be seen as a positive contribution to the society by combating cyber threats and protecting individuals' data and privacy.

However, there are also social considerations that need to be addressed when implementing such a system. For instance, issues related to privacy and data protection may arise, as the detection of XSS involves analyzing user data and online activities. It is important to ensure that the system complies with relevant regulations and guidelines to safeguard user privacy and prevent any misuse of personal information.

Moreover, there may be concerns about the potential impact of the proposed solution on user experience and internet freedom. Users may be apprehensive about the level of surveillance and monitoring involved in XSS detection, and there might be resistance to adopting new technologies that could disrupt the way they interact with online platforms.

To address these social feasibility challenges, it is essential to engage with stakeholders, including users, privacy advocates, policymakers, and industry experts, throughout the development and implementation process. Transparent communication, ethical considerations, and user-friendly design are key elements for ensuring the acceptance and support of the CNN and transformer-based XSS detection system within the society. By proactively addressing social concerns and seeking input from relevant parties, the implementation of such a solution can be made socially feasible and beneficial for all stakeholders.

3.4 REQUIREMENT SPECIFICATION

3.4.1 HARDWARE REQUIREMENTS

Hard disk	: 120 GB
RAM	: 2GB (minimum)
Keyboard	: 110 keys enhanced

3.4.2 SOFTWARE REQUIREMENTS

Operating system	: MacOS
Language	: Python

3.5 LANGUAGE SPECIFICATION - PYTHON

Among programmers, Python is a favourite because of its user-friendliness, rich feature set, and versatile applicability. Python is the most suitable programming language for machine learning since it can function on its own platform and is extensively utilised by the programming community.

Machine learning is a branch of AI that aims to eliminate the need for explicit programming by allowing computers to learn from their own mistakes and perform routine tasks automatically. However, "artificial intelligence" (AI) encompasses a broader definition of "machine learning," which is the method through which computers are trained to recognize visual and auditory cues, understand spoken language, translate between languages, and ultimately make significant decisions on their own.

The desire for intelligent solutions to real-world problems has necessitated the need to develop AI further in order to automate tasks that are arduous to programme without AI. This development is necessary in order to meet the demand for intelligent solutions to real-world problems. Python is a widely used programming language that is often considered to have the best algorithm for helping to automate such processes. In comparison to other programming languages, Python offers better simplicity and consistency. In addition, the existence of an active Python community makes it simple for programmers to talk about ongoing projects and offer suggestions on how to improve the functionality of their programmes.

ADVANTAGES OF USING PYTHON

Following are the advantages of using Python:

- Variety of Framework and libraries:

A good programming environment requires libraries and frameworks. Python frameworks and libraries simplify programme development. Developers can speed up complex project coding with prewritten code from a library. PyBrain, a modular machine learning toolkit in Python, provides easy-to-use algorithms. Python frameworks and libraries provide a structured and tested environment for the best coding solutions.

- Reliability

Most software developers seek simplicity and consistency in Python. Python code is concise and readable, simplifying presentation. Compared to other programming languages, developers can write code quickly. Developers can get community feedback to improve their product or app. Python is simpler than other programming languages, therefore beginners may learn it quickly. Experienced developers may focus on innovation and solving real-world problems with machine learning because they can easily design stable and trustworthy solutions.

- Easily Executable

Developers choose Python because it works on many platforms without change. Python runs unmodified on Windows, Linux, and macOS. Python is supported on all these platforms, therefore you don't need a Python expert to comprehend it. Python's great executability allows separate applications. Programming the app requires only Python. Developers benefit from this because some programming languages require others to complete the job. Python's portability cuts project execution time and effort.

CHAPTER 4
SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

System architecture refers to the conceptual design of a computer system, encompassing its structure, components, interfaces, and behavior. It defines the overall layout and connectivity of the system, determining how different parts interact and function together to achieve the desired objectives. A well-designed system architecture ensures scalability, performance, and reliability of the system while meeting the specified requirements. It serves as a blueprint for developers to understand the system's organization and make informed decisions regarding technology choices, data flow, and communication protocols. Effective system architecture plays a crucial role in the success of a project by providing a solid foundation for development and implementation.

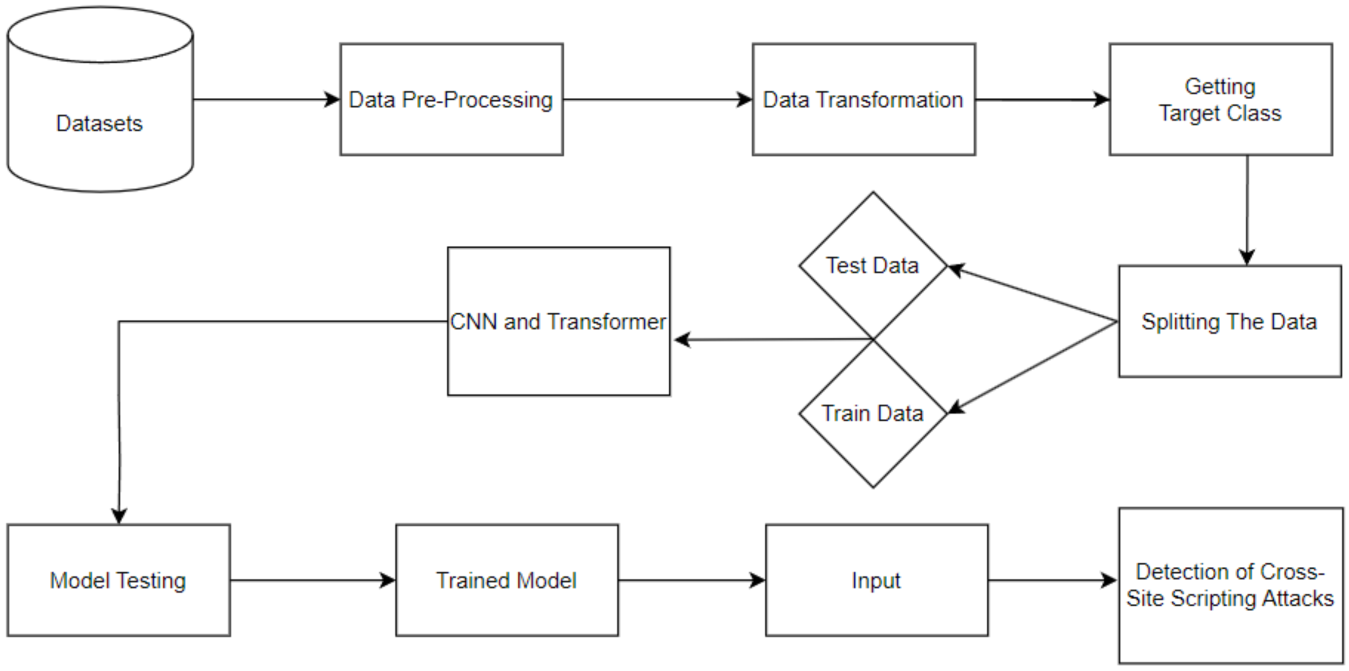


Fig 4.1 SYSTEM ARCHITECTURE

4.7 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a visual representation of how data flows within a system. It uses symbols to depict the processes that manipulate data, the data stores where information is kept, and the data flows that show how data moves between processes and stores. DFDs help stakeholders understand the system's overall functionality, identify potential bottlenecks, and ensure that data is accessible and secure. They are an essential tool for systems analysis and design, enabling teams to create and optimize efficient data processes. By mapping out data flow, stakeholders can better understand the system's architecture and make informed decisions for improvement.

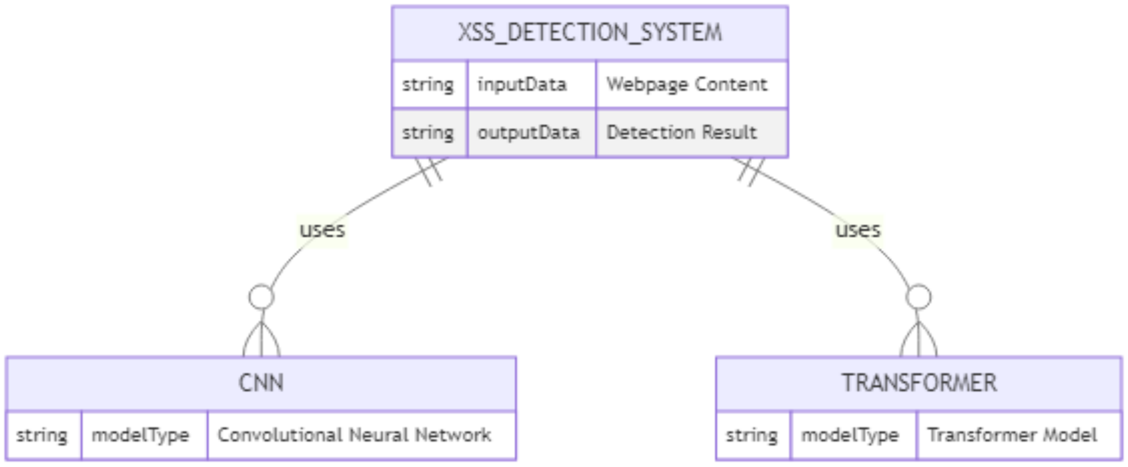


Fig 4.2 DATA FLOW DIAGRAM Level 1

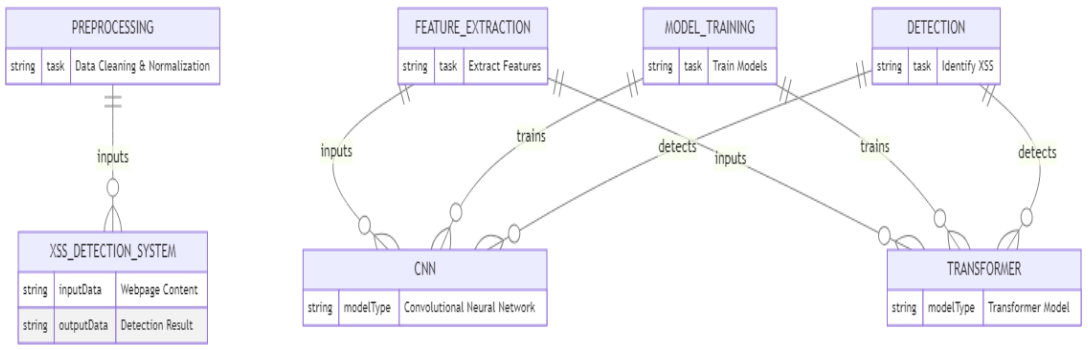


Fig 4.3 DATA FLOW DIAGRAM Level 2

CHAPTER 5
MODULE DESCRIPTION

5.1 Data Collection and Formatting

1. Data Collection and Formatting

In the context of detecting cross-site scripting (XSS) using convolutional neural networks (CNN) and transformers, the first step involves collecting a diverse dataset of XSS and non-XSS payloads from various sources. This dataset should be carefully curated to ensure a balanced representation of both malicious and benign inputs. Next, the data must be preprocessed and formatted in a way that is compatible with the input requirements of the CNN and transformer models. This may involve tokenization, padding, and encoding the payloads into numerical representations. Additionally, any necessary data augmentation techniques should be applied to enhance the model's ability to generalize.

2. Feature Engineering and Extraction

Feature Engineering and Extraction play a crucial role in the detection of cross-site scripting using Convolutional Neural Networks (CNNs) and transformers. Feature engineering involves selecting the most relevant features from the data to improve the model's performance, while feature extraction involves transforming the input data into a more suitable representation for the model. In the context of detecting cross-site scripting, these techniques can help in identifying patterns and characteristics of malicious code. By extracting features such as code snippets, HTML tags, and other relevant information from the input data, the model can learn to differentiate between legitimate and malicious scripts. Furthermore, the use of CNNs and transformers allows for the efficient processing of these complex features and patterns, enabling the detection of cross-site scripting attacks with high accuracy and reliability.

3. Text Tokenization and Embedding

Text tokenization and embedding play a crucial role in the detection of cross-site scripting through the utilization of Convolutional Neural Networks (CNNs) and Transformers. Tokenization involves breaking down text into individual tokens or words, which forms the foundation for subsequent analysis. Embedding refers to the transformation of tokens into dense, numerical representations that capture semantic relationships between words. By employing CNNs, the model can effectively learn hierarchical features from tokenized text data, enabling the detection of patterns indicative of cross-site scripting attacks. Integrating transformers enhances the model's ability to capture long-range dependencies and contextual information within the text, leading to more nuanced detection capabilities. The combination of tokenization and embedding techniques with CNNs and transformers offers a powerful approach for accurately identifying and mitigating cross-site scripting vulnerabilities.

4. Model Training with CNN

In the task of detecting cross-site scripting (XSS) with a focus on utilizing convolutional neural networks (CNN) and transformers in the model training process, a comprehensive approach can be developed to effectively identify and prevent XSS attacks. By leveraging the capabilities of CNN for feature extraction and pattern recognition, along with the power of transformers for capturing long-range dependencies and context, the model can learn intricate relationships within the input data. The CNN can learn spatial hierarchies of features, while transformers can attend to relevant parts of the input sequence. By combining these two architectures in the training process, the model can enhance its ability to detect XSS patterns efficiently. Through effective training strategies and optimization techniques, such as data augmentation and fine-tuning, the CNN with transformers model can achieve robust performance in identifying and mitigating XSS vulnerabilities.

5. Fine-Tuning with Transformers

Fine-tuning the Transformer model with Convolutional Neural Networks (CNN) for the detection of cross-site scripting involves incorporating the strengths of both architectures. By leveraging the attention mechanism in Transformers for capturing long-range dependencies and the ability of CNNs to learn spatial hierarchies, the model can effectively identify XSS vulnerabilities in web applications. The Transformer's self-attention mechanism enables the model to focus on relevant tokens, while the CNN layers provide the spatial filtering necessary to extract meaningful features from the input data. Through fine-tuning, the model can learn to distinguish between legitimate and malicious code snippets, ultimately enhancing the detection capabilities of XSS attacks. This hybrid approach combining Transformers and CNNs showcases promising results in improving the accuracy and robustness of XSS detection systems.

5.2 MODEL IMPROVISATION

1. Overview of Cross-site Scripting (XSS)

Cross-site scripting (XSS) is a type of cyber attack where malicious scripts are injected into web pages viewed by unsuspecting users. To detect XSS using Convolutional Neural Networks (CNNs) and transformers, the approach involves leveraging the capabilities of deep learning models to analyze and identify patterns in web content that could potentially be indicative of XSS attacks. By training CNNs and transformers on large datasets of web content, the models can learn to distinguish between legitimate code and potentially malicious script injections. Through this process, the detection of XSS attacks can be automated and improved, providing better security for web applications and protecting users from potential vulnerabilities.

2. Introduction to Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are a type of deep learning model widely used in image recognition tasks due to their ability to automatically learn features from data. In the context of detecting cross-site scripting (XSS) attacks, CNNs can be leveraged to process input data and extract relevant patterns indicative of malicious script code. By training a CNN on a labeled dataset of XSS and non-XSS instances, the network learns to differentiate between the two categories based on patterns in the input data. Additionally, transformers, which are powerful models based on attention mechanisms, can be incorporated to enhance the performance of the CNN by capturing long-range dependencies in the input sequences. By combining CNNs and transformers, it is possible to create a robust and efficient system for detecting XSS attacks in web applications, leveraging the strengths of both models to improve the accuracy and effectiveness of the detection process.

3. Application of Transformers in XSS Detection

One key application of transformers in XSS detection is the utilization of Convolutional

Neural Networks (CNN) alongside transformers for more accurate and effective detection of cross-site scripting attacks. Transformers excel in capturing long-range dependencies in sequences, enabling them to analyze the context of the code more effectively. By integrating transformers with CNNs, the model can leverage the strengths of both architectures, with CNNs focusing on local features and transformers capturing global relationships in the code. This combined approach enhances the detection capabilities of the system, allowing it to identify complex XSS patterns embedded within the code. Furthermore, transformers can adapt to diverse input formats and structures, making them versatile in processing varied kinds of XSS attack vectors. The integration of transformers with CNNs in XSS detection represents a promising advancement in enhancing the security measures against such cyber threats.

5.3 CREATING USER INTERFACE

Web User Interface

Cross-site scripting (XSS) is a type of cyber attack where malicious scripts are injected into web pages viewed by unsuspecting users. To detect XSS using Convolutional Neural Networks (CNNs) and transformers, the approach involves leveraging the capabilities of deep learning models to analyze and identify patterns in web content that could potentially be indicative of XSS attacks. By training CNNs and transformers on large datasets of web content, the models can learn to distinguish between legitimate code and potentially malicious script injections. Through this process, the detection of XSS attacks can be automated and improved, providing better security for web applications and protecting users from potential vulnerabilities.

Database

Convolutional Neural Networks (CNN) are a type of deep learning model widely used in image recognition tasks due to their ability to automatically learn features from data. In the context of detecting cross-site scripting (XSS) attacks, CNNs can be leveraged to process input data and extract relevant patterns indicative of malicious script code. By training a CNN on a labeled dataset of XSS and non-XSS instances, the network learns to differentiate between the two categories based on patterns in the input data. Additionally, transformers, which are powerful models based on attention mechanisms, can be incorporated to enhance the performance of the CNN by capturing long-range dependencies in the input sequences. By combining CNNs and transformers, it is possible to create a robust and efficient system for detecting XSS attacks in web applications, leveraging the strengths of both models to improve the accuracy and effectiveness of the detection process.

Security

One key application of transformers in XSS detection is the utilization of Convolutional Neural Networks (CNN) alongside transformers for more accurate and effective detection of cross-site scripting attacks. Transformers excel in capturing long-range dependencies in sequences, enabling them to analyze the context of the code more effectively. By integrating transformers with CNNs, the model can leverage the strengths of both architectures, with CNNs focusing on local features and transformers capturing global relationships in the code. This combined approach enhances the detection capabilities of the system, allowing it to identify complex XSS patterns embedded within the code. Furthermore, transformers can adapt to diverse input formats and structures, making them versatile in processing varied kinds of XSS attack vectors. The integration of transformers with CNNs in XSS detection represents a promising advancement in enhancing the security measures against such cyber threats.

CHAPTER 6
TESTING

Discovering and fixing such problems is what testing is all about. The purpose of testing is to find and correct any problems with the final product. It's a method for evaluating the quality of the operation of anything from a whole product to a single component. The goal of stress testing software is to verify that it retains its original functionality under extreme circumstances. There are several different tests from which to pick. Many tests are available since there is such a vast range of assessment options.

Who Performs the Testing: All individuals who play an integral role in the software development process are responsible for performing the testing. Testing the software is the responsibility of a wide variety of specialists, including the End Users, Project Manager, Software Tester, and Software Developer.

When it is recommended that testing begin: Testing the software is the initial step in the process. begins with the phase of requirement collecting, also known as the Planning phase, and ends with the stage known as the Deployment phase. In the waterfall model, the phase of testing is where testing is explicitly arranged and carried out. Testing in the incremental model is carried out at the conclusion of each increment or iteration, and the entire application is examined in the final test.

When it is appropriate to halt testing: Testing the programme is an ongoing activity that will never end. Without first putting the software through its paces, it is impossible for anyone to guarantee that it is completely devoid of errors. Because the domain to which the input belongs is so expansive, we are unable to check every single input.

6.1 TYPES OF TESTING

UNIT TESTING

Unit testing for the detection of cross-site scripting (XSS) using Convolutional Neural Networks (CNNs) and transformers involves verifying the individual components and functions of the algorithms for identifying XSS vulnerabilities in web applications.

Three test cases can be designed for unit testing this functionality:

Testcase1: Input a benign script into the CNN model and confirm that the output is classified as non-XSS.
Testcase2: Provide a malicious XSS script to the transformer model and validate that the output correctly flags it as a potential XSS attack.
Testcase3: Evaluate the combined CNN and transformer model with a mix of benign and malicious scripts to ensure accurate detection and classification of XSS vulnerabilities.

These test cases aim to validate the effectiveness and accuracy of the XSS detection system, ensuring that it can differentiate between safe and malicious scripts effectively.

INTEGRATION TESTING

Integration testing is essential for detecting cross-site scripting (XSS) vulnerabilities in systems that utilize Convolutional Neural Networks (CNN) and transformers for data processing. During integration testing, the focus is on ensuring seamless interaction between various components, such as the CNN model, transformer module, and input data to prevent XSS attacks.

Testcase1: Input Validation - Validate that input data passed through the transformer module is properly sanitized to prevent XSS vulnerabilities by including special characters or scripts.
Testcase2: Output Encoding - Verify that the output generated by the CNN model is encoded to prevent script injection and that it is safely displayed on the user interface.
Testcase3: Integration Flow - Test the end-to-end flow of data processing, ensuring that data passed between the CNN, transformer, and user interface layers is secure and protected against XSS attacks.

By implementing these test cases in integration testing, the system's robustness against

XSS vulnerabilities can be significantly enhanced when using CNNs and transformers.

FUNCTIONAL TESTING

Functional testing for the detection of cross-site scripting (XSS) using convolutional neural networks (CNN) and transformers involves validating the effectiveness of the machine learning models in identifying and preventing XSS vulnerabilities in web applications. The testing process involves providing input data containing potential XSS attacks to the models and analyzing their outputs to ensure accurate detection and mitigation of XSS threats.

Testcase1: Input a script with <script>alert('XSS')</script> into the model and verify that it correctly recognizes and flags it as a potential XSS attack.

Testcase2: Enter a URL containing a malicious JavaScript code into the model and confirm that it identifies the XSS vulnerability.

Testcase3: Input a combination of HTML and JavaScript code designed to bypass traditional XSS filters and assess the model's ability to detect and neutralize the threat.

By conducting these test cases and evaluating the performance of the CNN and transformer models, the effectiveness of using machine learning for XSS detection can be assessed and any necessary improvements can be implemented.

BLACK BOX TESTING

Black box testing for the detection of cross-site scripting using Convolutional Neural Networks (CNN) and transformers involves analyzing the input and output behaviors of a web application without any knowledge of its internal code structure. This approach evaluates the application's security by feeding various inputs and checking for potential XSS vulnerabilities based on the response.

Testcase1: Input a string containing a script tag <script>alert('XSS');</script> and check if the output filters out the script tag to prevent execution.

Testcase2: Input a URL parameter with JavaScript code appended to it, such as ?

search=<script>alert('XSS')</script>, and verify that the application sanitizes the URL parameter to prevent XSS attacks.

Testcase3: Input a malicious payload disguised within a legitimate data entry, like <img src=x onerror="alert('XSS')">, and confirm that the application detects and neutralizes the payload before rendering it on the webpage.

WHITE BOX TESTING

White box testing is a method where the internal structure of the software application is known and tested for vulnerabilities. In the context of detecting cross-site scripting (XSS) vulnerabilities, using Convolutional Neural Networks (CNN) and transformers can be effective in identifying malicious code injection within web applications.

In this approach, the CNN can analyze the code structure and patterns, while transformers can help in understanding contextual information for accurate detection of XSS. Through white box testing, these models can be trained on known XSS attack vectors to highlight potential vulnerabilities in the application code.

Testcase1: Injecting a script tag with malicious code into a text input field to check if the CNN and transformers can correctly flag it as an XSS vulnerability.

Testcase2: Attempting to bypass input validation by encoding special characters within a URL parameter and verifying if the detection models can identify this as a potential XSS threat.

Testcase3: Embedding JavaScript code within a hidden form field and submitting the form to assess if the testing tools can flag this as a security risk.

CHAPTER 7
CONCLUSION

7.1 CONCLUSION

In conclusion, the utilization of Convolutional Neural Networks (CNN) and transformers for the detection of cross-site scripting (XSS) represents a promising approach to addressing the evolving challenges posed by malicious web attacks. By harnessing the power of CNN to extract salient features from web content and combining it with the sequence-to-sequence learning capabilities of transformers, researchers and security professionals can enhance the accuracy and efficiency of XSS detection mechanisms. The fusion of these deep learning architectures enables the model to effectively learn complex patterns and relationships within web data, thereby improving the ability to identify and mitigate XSS attacks in real-time. Moreover, the adaptability and scalability afforded by this hybrid approach make it well-suited to the dynamic nature of modern web environments, where new attack vectors continuously emerge. As the threat landscape evolves and cyber attackers become more sophisticated, the integration of CNN and transformers in XSS detection systems offers a robust and intelligent solution for safeguarding web applications and protecting user data from malicious exploitation. Through ongoing research and refinement, the synergy between CNN and transformers holds significant potential for enhancing the resilience of web security measures against XSS vulnerabilities.

CHAPTER 8
APPENDIX 1
TRANSFORMER MODEL IMPLEMENTATION

```
import numpy as np
import pandas as pd
import torch
from tqdm import tqdm
import string
df = pd.read_csv("XSS_dataset.csv", index_col=0)
df.head()
# split train and test
train_df = df[: int(len(df) * 0.8)]
test_df = df[int(len(df) * 0.8) :]
train_df
def find_character(data, max_len):
    alphabet = string.ascii_lowercase + string.digits + string.punctuation + " "
    alphabet
    mat = []
    for ch in data:
        if ch not in alphabet:
            continue
        mat.append(alphabet.index(ch))
    if len(mat) < max_len:
        mat += [0] * (max_len - len(mat))
    elif len(mat) > max_len:
        mat = mat[:max_len]
    return mat
class Dataset(torch.utils.data.Dataset):
    def __init__(self, df, max_len) -> None:
        self.df = df
        self.max_len = max_len
    def __len__(self):
        return len(self.df)
    def __getitem__(self, index):
        sentence = self.df["Sentence"].values[index]
        label = self.df["Label"].values[index]
        return torch.tensor(find_character(sentence, self.max_len)), torch.tensor(label)
trainDataset = Dataset(train_df, 1000)
testDataset = Dataset(test_df, 1000)
len(trainDataset), len(testDataset)
trainGenerator = torch.utils.data.DataLoader(trainDataset, batch_size=128, shuffle=True)
testGenerator = torch.utils.data.DataLoader(testDataset, batch_size=128, shuffle=True)
for data, label in trainGenerator:
    print(data.shape)
    print(label.shape)
    break
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import TransformerEncoder, TransformerEncoderLayer
import math
class CustomTransformerModel(nn.Module):
    def __init__(
        self,
        vocab_size,
        embedding_size,
        num_classes,
        nhead,
        num_layers,
        dim_feedforward,
        dropout,
    ):
        super(CustomTransformerModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.pos_encoder = PositionalEncoding(embedding_size, dropout)
        encoder_layers = TransformerEncoderLayer(
            embedding_size, nhead, dim_feedforward, dropout
        )
        self.transformer_encoder = TransformerEncoder(encoder_layers, num_layers)
        self.fc = nn.Linear(embedding_size, num_classes)
        self.init_weights()
    def init_weights(self):
        initrange = 0.1
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()
        self.fc.weight.data.uniform_(-initrange, initrange)
    def forward(self, x):
        embedded = self.embedding(x)
        embedded = embedded.permute(
            1, 0, 2
        )  # Shape: [sequence_length, batch_size, embedding_size]
        embedded = self.pos_encoder(embedded)
        output = self.transformer_encoder(embedded)
        output = output.mean(dim=0)  # Average pooling over the sequence length
        output = self.fc(output)
        return output
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer("pe", pe)
    def forward(self, x):
        x = x + self.pe[: x.size(0), :]
        return self.dropout(x)
# Example usage
model = CustomTransformerModel(
    vocab_size=70,
    embedding_size=64,
    num_classes=2,
    nhead=8,
    num_layers=3,
    dim_feedforward=128,
    dropout=0.1,
)
output = model(torch.tensor([1, 2, 3, 4, 5]).unsqueeze(1))  # Example input tensor
print(output)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
loss = torch.nn.CrossEntropyLoss()
epochs = 10
# Check if CUDA is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# Move the model to the appropriate device
model.to(device)
for epoch in range(epochs):
    model.train()
    for data, label in tqdm(trainGenerator):
        data = data.to(device)
        label = label.to(device)
        optimizer.zero_grad()
        output = model(data)
        l = loss(output, label)
        l.backward()
        optimizer.step()
    print(f"epoch: {epoch}, loss: {l}", end="")
    # eval
    model.eval()
    right_num = 0
    for data, label in testGenerator:
        data = data.to(device)
        label = label.to(device)
        output = model(data)
        right_num += (torch.argmax(output, dim=1) == label).sum().item()
    print(f"\nTest accuracy: {right_num / len(testDataset)}")
class EvaluationMetrics:
    def __init__(self):
        self.true_positive = 0
        self.false_positive = 0
        self.false_negative = 0
        self.true_negative = 0
    def add(self, prediction, label):
        if prediction == 1 and label == 1:
            self.true_positive += 1
        elif prediction == 1 and label == 0:
            self.false_positive += 1
        elif prediction == 0 and label == 1:
            self.false_negative += 1
        elif prediction == 0 and label == 0:
            self.true_negative += 1
    def accuracy(self):
        return (self.true_positive + self.true_negative) / (
            self.true_positive
            + self.false_positive
            + self.false_negative
            + self.true_negative
        )
    def precision(self):
        return self.true_positive / (self.true_positive + self.false_positive)
    def recall(self):
        return self.true_positive / (self.true_positive + self.false_negative)
# Check if CUDA is available
if torch.cuda.is_available():
    device = torch.device("cuda")
    print(
        f"Using CUDA device: {torch.cuda.get_device_name(torch.cuda.current_device())}"
    )
else:
    device = torch.device("cpu")
    print("CUDA is not available. Using CPU.")
# Move model to the selected device
model.to(device)
# Set model to evaluation mode
model.eval()
evaluation = EvaluationMetrics()
for data, label in testGenerator:
    # Move data to the selected device
    data = data.to(device)
    label = label.to(device)
    # Perform inference
    output = model(data).argmax(dim=1)
    # Update evaluation metrics
    for pred, true_label in zip(output, label):
        evaluation.add(pred.item(), true_label.item())
# Calculate and print evaluation metrics
print(f"Accuracy: {evaluation.accuracy()}")
print(f"Precision: {evaluation.precision()}")
print(f"Recall: {evaluation.recall()}")
def predict(sentence):
    model.eval()
    with torch.no_grad():
        data = torch.tensor(find_character(sentence, 1000)).unsqueeze(0).to(device)
        output = model(data).argmax(dim=1)
        return output.item()
import string
def find_character(data, max_len):
    alphabet = string.ascii_lowercase + string.digits + string.punctuation + " "
    alphabet
    mat = []
    for ch in data:
        if ch not in alphabet:
            continue
        mat.append(alphabet.index(ch))
    if len(mat) < max_len:
        mat += [0] * (max_len - len(mat))
    elif len(mat) > max_len:
        mat = mat[:max_len]
    return mat
class CustomTransformerModel(nn.Module):
    def __init__(
        self,
        vocab_size,
        embedding_size,
        num_classes,
        nhead,
        num_layers,
        dim_feedforward,
        dropout,
    ):
        super(CustomTransformerModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.pos_encoder = PositionalEncoding(embedding_size, dropout)
        encoder_layers = TransformerEncoderLayer(
            embedding_size, nhead, dim_feedforward, dropout
        )
        self.transformer_encoder = TransformerEncoder(encoder_layers, num_layers)
        self.fc = nn.Linear(embedding_size, num_classes)
        self.init_weights()
    def init_weights(self):
        initrange = 0.1
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()
        self.fc.weight.data.uniform_(-initrange, initrange)
    def forward(self, x):
        embedded = self.embedding(x)
        embedded = embedded.permute(
            1, 0, 2
        )  # Shape: [sequence_length, batch_size, embedding_size]
        embedded = self.pos_encoder(embedded)
        output = self.transformer_encoder(embedded)
        output = output.mean(dim=0)  # Average pooling over the sequence length
        output = self.fc(output)
        return output
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer("pe", pe)
    def forward(self, x):
        x = x + self.pe[: x.size(0), :]
        return self.dropout(x)
# Example usage
loaded_model = CustomTransformerModel(
    vocab_size=70,
    embedding_size=64,
    num_classes=2,
    nhead=8,
    num_layers=3,
    dim_feedforward=128,
    dropout=0.1,
)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
loaded_model.load_state_dict(torch.load("model.pth", map_location=device))
def predictLoaded(sentence):
    loaded_model.eval()
    with torch.no_grad():
        data = torch.tensor(find_character(sentence, 1000)).unsqueeze(0).to(device)
        output = loaded_model(data).argmax(dim=1)
        return output.item()
predictLoaded(sentence="<details onpointerover=alert(1)>XSS</details>")
```

```
import matplotlib as plt
import itertools
def plot_confusion_matrix(
    cm, classes, normalize=False, title="Confusion matrix", cmap=plt.cm.Blues
):
    """Plots the confusion matrix."""
    if normalize:
        cm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print("Confusion matrix, without normalization")
    plt.imshow(cm, interpolation="nearest", cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=55)
    plt.yticks(tick_marks, classes)
    fmt = ".2f" if normalize else "d"
    thresh = cm.max() / 2.0
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(
            j,
            i,
            format(cm[i, j], fmt),
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black",
        )
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
    plt.tight_layout()
device = 'cpu'
from sklearn.metrics import classification_report, confusion_matrix
y_pred = []
y_true = []
for data, label in testGenerator:
    # Move data to the selected device
    data = data.to(device)
    label = label.to(device)
    # Perform inference
    output = model(data).argmax(dim=1)
    # Update evaluation metrics
    for pred, true in zip(output, label):
        y_pred.append(pred)
        y_true.append(true)
class_names = ["XSS", "Normal"]
print(classification_report(y_true, y_pred, target_names=class_names))
cm = confusion_matrix(y_true, y_pred)
plot_confusion_matrix(
    cm, classes=class_names, title=f"Confusion Matrix for ResNet50"
)
```

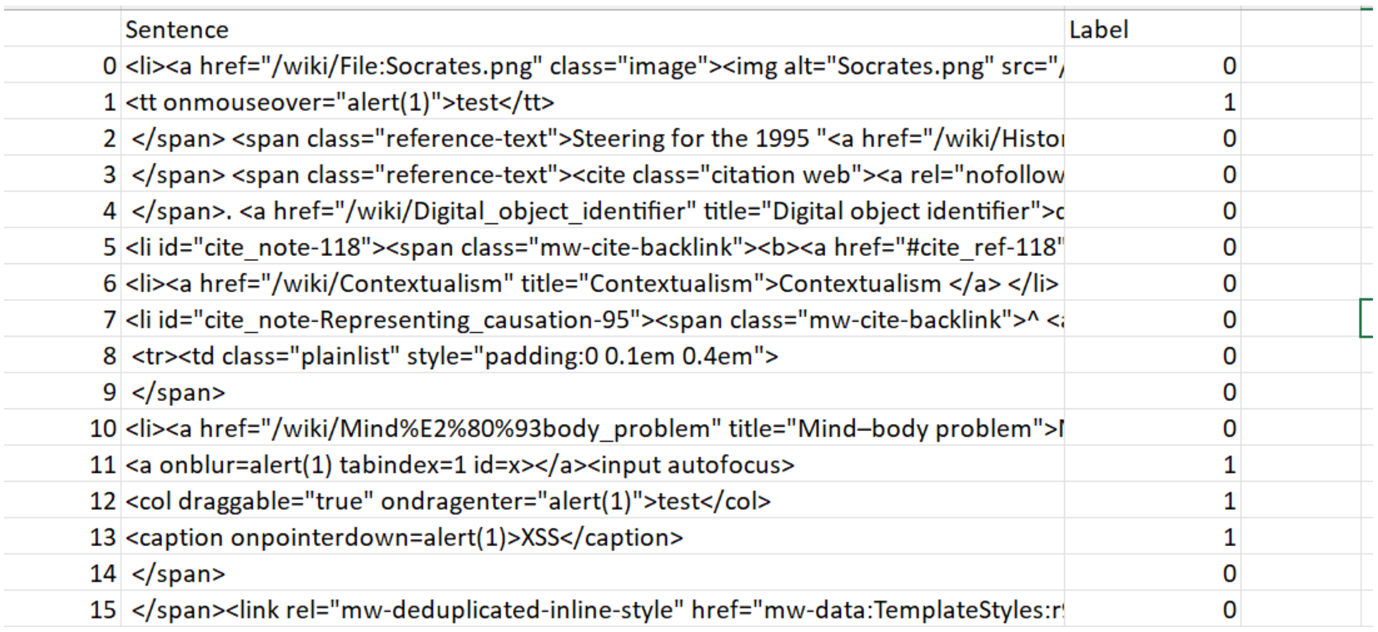
CNN MODEL IMPLEMENTATION

```
import numpy as np
import pandas as pd
import glob
import time
import pandas as pd

import os
import matplotlib.pyplot as plt
import keras

import cv2



def convert_to_ascii(sentence):
    sentence_ascii=[]

    for i in sentence:


        """Some characters have values very big e.d 8221 adn some are chinese letters
        I am removing letters having values greater than 8222 and for rest greater
        than 128 and smaller than 8222 assigning them values so they can easily be normalized"""

        if(ord(i)<8222):

            if(ord(i)==8217): # ’  :  8217
                sentence_ascii.append(134)


            if(ord(i)==8221): # ”  :  8221
                sentence_ascii.append(129)

            if(ord(i)==8220): # “  :  8220
                sentence_ascii.append(130)


            if(ord(i)==8216): # ‘  :  8216
                sentence_ascii.append(131)
            if(ord(i)==8217): # ’  :  8217
               sentence_ascii.append(132)

            if(ord(i)==8211): # –  :  8211
               sentence_ascii.append(133)


            """
            If values less than 128 store them else discard them
            """
            if (ord(i)<=128):
                  sentence_ascii.append(ord(i))

            else:
                  pass


      zer=np.zeros((10000))

      for i in range(len(sentence_ascii)):
         zer[i]=sentence_ascii[i]

      zer.shape=(100, 100)



      return zer

   arr=np.zeros((len(sentences),100,100))

   for i in range(len(sentences)):

      image=convert_to_ascii(sentences[i])

      x=np.asarray(image,dtype='float')
      image =  cv2.resize(x, dsize=(100,100), interpolation=cv2.INTER_CUBIC)
      image/=128

      arr[i]=image


   # A basic CNN Model
   # Number of layers = 11
   # Number of Convolutional layer: 3

   model=tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(64,(3,3), activation=tf.nn.relu, input_shape=(100,100,1)),
      tf.keras.layers.MaxPooling2D(2,2),

      tf.keras.layers.Conv2D(128,(3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2,2),

      tf.keras.layers.Conv2D(256,(3,3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2,2),

      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(256, activation='relu'),
      tf.keras.layers.Dense(128, activation='relu'),
      tf.keras.layers.Dense(64, activation='relu'),
      tf.keras.layers.Dense(1, activation='sigmoid')

   ])


   model.compile(loss='binary_crossentropy',
                 optimizer='adam',
                 metrics=['accuracy'])
   model.summary()


   # Stop when validation accuracy > 97

   class myCallback(tf.keras.callbacks.Callback):
      def on_epoch_end(self, epoch, logs={}):
         if(logs.get('val_accuracy')>0.97):
            print(" \n Reached 97% + validation accuracy")
            self.model.stop_training=True
   callbacks = myCallback()

   # Install necessary libraries if not already installed
   !pip install pandas numpy matplotlib seaborn

   import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   import seaborn as sns

   # Load the dataset from CSV
   data = pd.read_csv("/content/XSS_dataset.csv")

   # Display basic information about the dataset
   print("Dataset Information:")
   print(data.info())
```

```
# Display first few rows of the dataset
print("\nFirst Few Rows of the Dataset:")
print(data.head())

# Check for missing values
missing_values = data.isnull().sum()
print("\nMissing Values:")
print(missing_values)

# Summary statistics for numerical features
print("\nSummary Statistics for Numerical Features:")
print(data.describe())

# Distribution of the target variable
plt.figure(figsize=(8, 6))
sns.countplot(data['Label'])
plt.title('Distribution of Target Variable (Label)')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()

# Visualize the distribution of HTML content lengths
data['Sentence_length'] = data['Sentence'].apply(len)
plt.figure(figsize=(10, 6))
sns.histplot(data['Sentence_length'], bins=30, kde=True)
plt.title('Distribution of HTML Content Lengths')
plt.xlabel('HTML Content Length')
plt.ylabel('Count')
plt.show()

from wordcloud import WordCloud

wordcloud = WordCloud(width=800, height=400, background_color='white').generate('
'.join(data['Sentence']))
plt.figure(figsize=(12, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of HTML Content')
plt.axis('off')
plt.show()
batch_size = 128
num_epoch = 10
#model training
model_log = model.fit(trainX, trainY,
        batch_size=batch_size,
        epochs=num_epoch,
        verbose=1,
        validation_data=( testX,  testY)
#                   callbacks=[callbacks]
                )
```

```
# prompt: confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
# Predict the labels for the test data
y_pred = model.predict(testX)
# Convert the predicted probabilities to binary labels
y_pred_binary = [int(round(p[0])) for p in y_pred]
# Create the confusion matrix
cm = confusion_matrix(testY, y_pred_binary)
# Define labels for the confusion matrix
labels = ['Non-XSS', 'XSS']
# Create a heatmap of the confusion matrix
sns.heatmap(cm, annot=True, fmt='d', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Validation Accuracy', 'Loss', 'Validation Loss', 'Precision', 'Recall'],
    'Value': [
        model_log.history['accuracy'][-1],
        model_log.history['val_accuracy'][-1],
        model_log.history['loss'][-1],
        model_log.history['val_loss'][-1],
        precision,
        recall
    ]
})
print(metrics_df.to_string())



# Further EDA as needed based on specific requirements
```

CHAPTER 9
APPENDIX 2



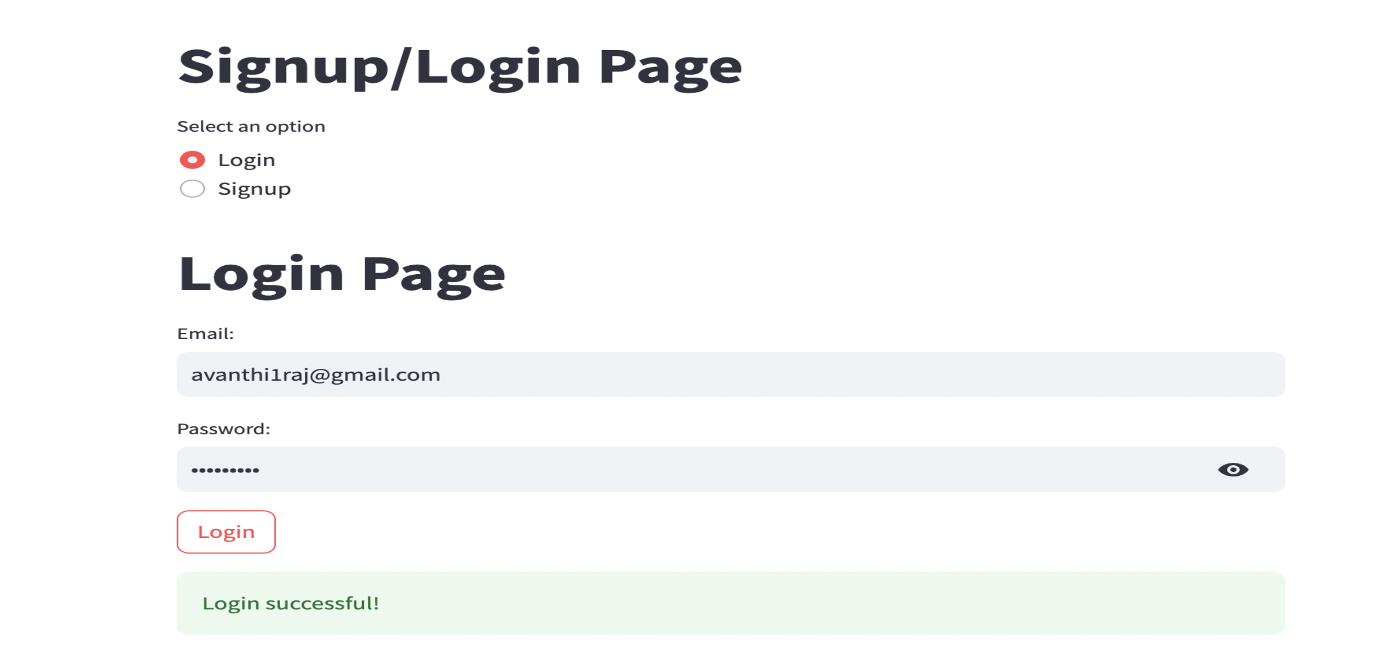
	Sentence	Label
0	`<li><a href="/wiki/File:Socrates.png" class="image"><img alt="Socrates.png" src="/`	0
1	`<tt onmouseover="alert(1)">test</tt>`	1
2	`</span> <span class="reference-text">Steering for the 1995 "<a href="/wiki/Histo`	0
3	`</span> <span class="reference-text"><cite class="citation web"><a rel="nofollow`	0
4	`</span>. <a href="/wiki/Digital_object_identifier" title="Digital object identifier">c`	0
5	`<li id="cite_note-118"><span class="mw-cite-backlink"><b><a href="#cite_ref-118"`	0
6	`<li><a href="/wiki/Contextualism" title="Contextualism">Contextualism </a> </li>`	0
7	`<li id="cite_note-Representing_causation-95"><span class="mw-cite-backlink">^ <`	0
8	`<tr><td class="plainlist" style="padding:0 0.1em 0.4em">`	0
9	`</span>`	0
10	`<li><a href="/wiki/Mind%E2%80%93body_problem" title="Mind–body problem">`	0
11	`<a onblur=alert(1) tabindex=1 id=x></a><input autofocus>`	1
12	`<col draggable="true" ondragenter="alert(1)">test</col>`	1
13	`<caption onpointerdown=alert(1)>XSS</caption>`	1
14	`</span>`	0
15	`</span><link rel="mw-deduplicated-inline-style" href="mw-data:TemplateStyles:r`	0

Fig 8.1. Dataset

	Accuracy	Precision	Recall	Loss
CNN	99.2%	99.7%	99.4%	0.023%
Transformers	99.4%	99.8%	99.0%	0.018%

Fig 8.2 Metrics Table

INPUT IMAGES:



Signup/Login Page

Select an option

[✓ Login] Signup

Login Page

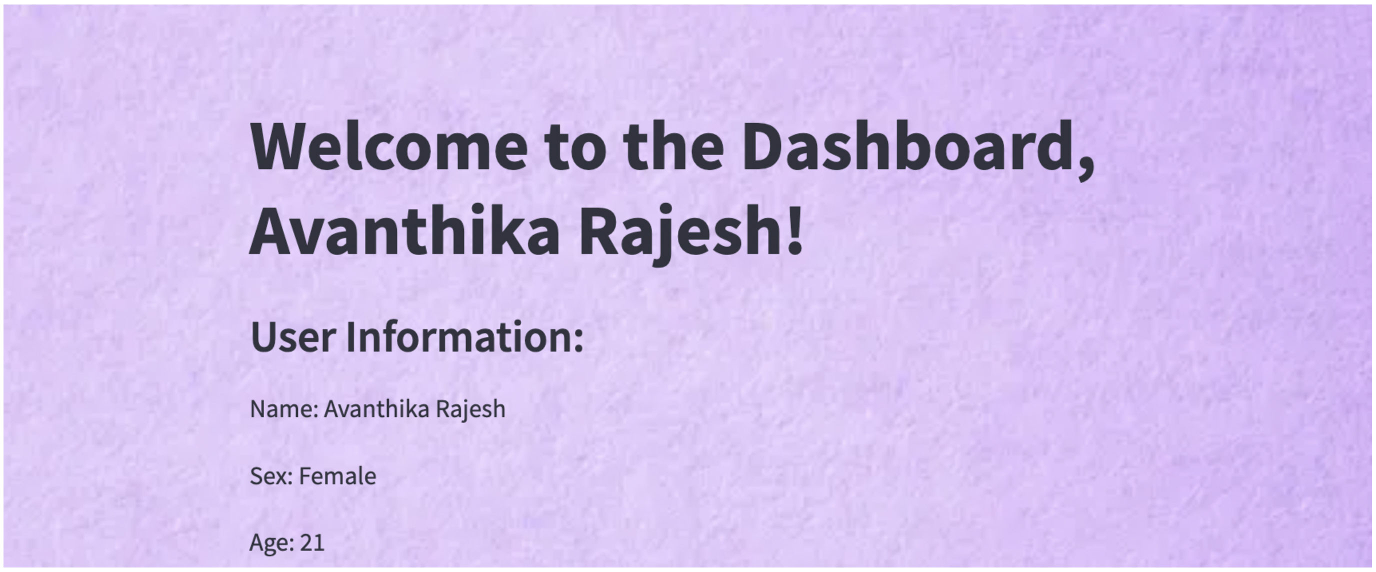
Email: avanthi1raj@gmail.com

Password: ••••••••••

Login

Login successful!

Fig 8.3. Sign-Up and Login Page



Welcome to the Dashboard, Avanthika Rajesh!

User Information:

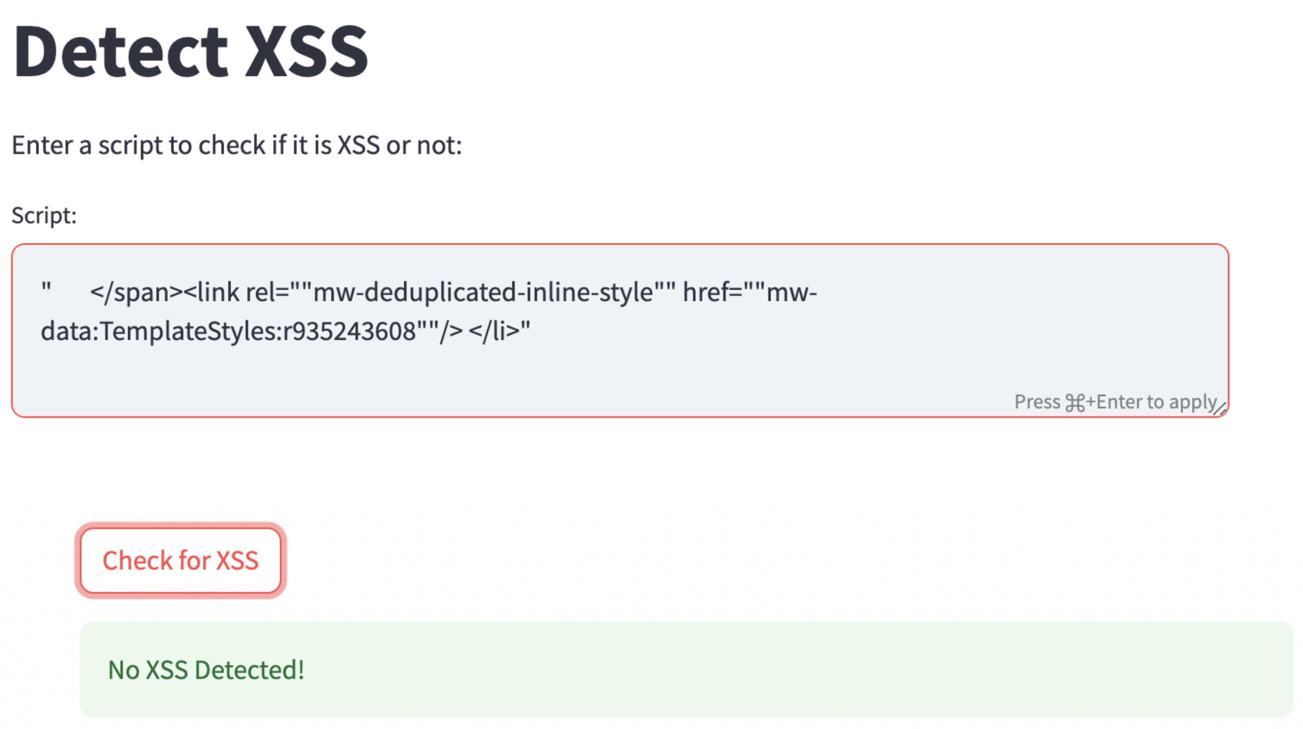
Name: Avanthika Rajesh

Sex: Female

Age: 21

Fig 8.4 . DashBoard

OUTPUT IMAGES


Detect XSS

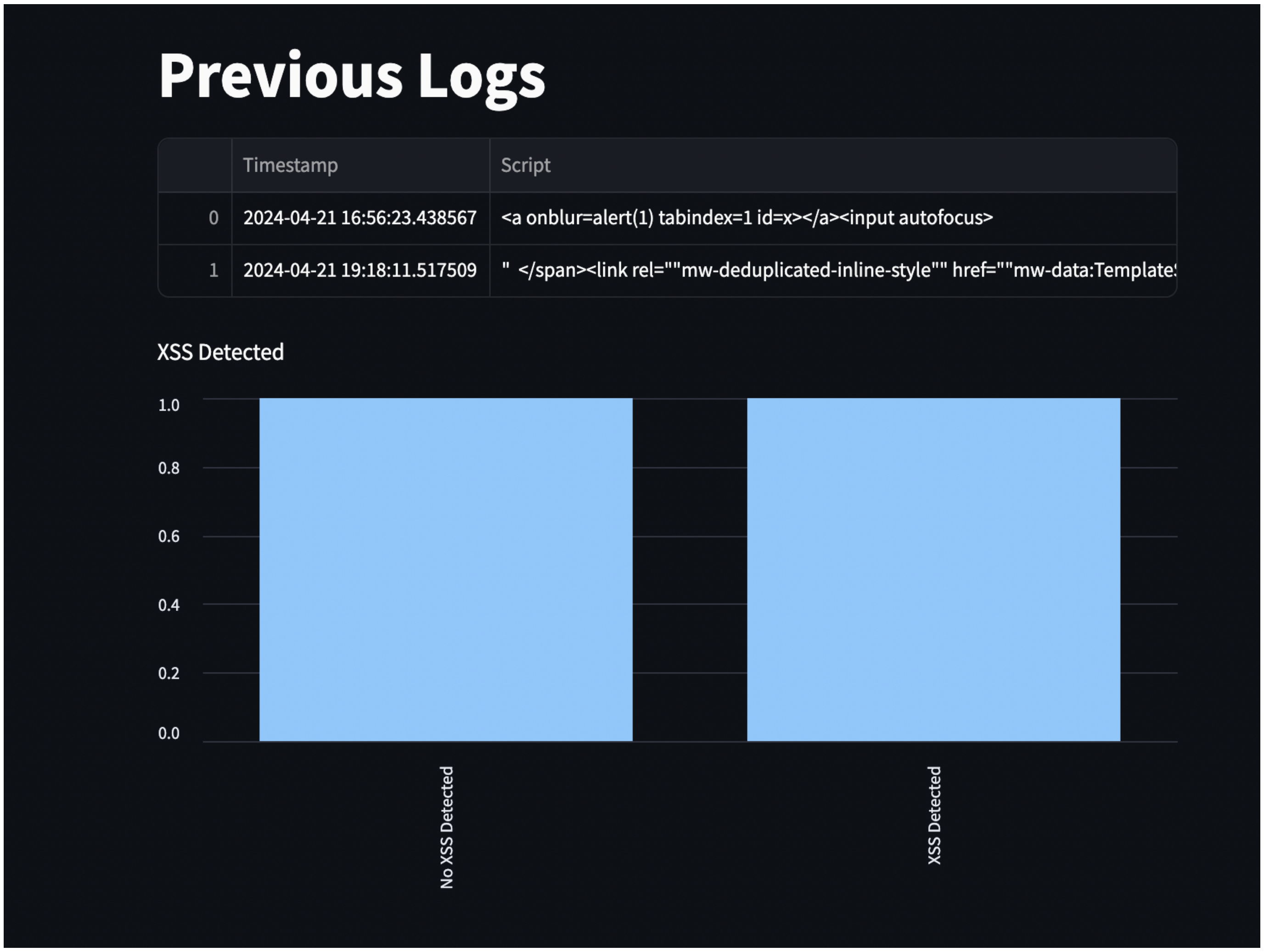
Enter a script to check if it is XSS or not:

Script:

```
"     </span><link rel=""mw-deduplicated-inline-style"" href=""mw-data:TemplateStyles:r935243608""/> </li>"
```

No XSS Detected!

Fig 8.5 . XSS Detection



Previous Logs

	Timestamp	Script
0	2024-04-21 16:56:23.438567	<a onblur=alert(1) tabindex=1 id=x></a><input autofocus>
1	2024-04-21 19:18:11.517509	" </span><link rel=""mw-deduplicated-inline-style"" href=""mw-data:Template

Fig 8.6 . XSS Logs

CHAPTER 10
REFERENCES

1. Kaur, J., Garg, U., & Bathla, G. (2023). Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. Artificial Intelligence Review, 1-45.

2. Al-Haija, Q. A. (2023). Cost-effective detection system of cross-site scripting attacks using hybrid learning approach. Results in Engineering, 19, 101266.

3. Chaudhary, P., Gupta, B. B., & Singh, A. K. (2023). Adaptive cross-site scripting attack detection framework for smart devices security using intelligent filters and attack ontology. Soft Computing, 27(8), 4593-4608.

4. Alaoui, R. L. (2023). Cross Site Scripting Attack Detection Approach Based on LSTM Encoder-Decoder and Word Embeddings. International Journal of Intelligent Systems and Applications in Engineering, 11(2), 277-282.

5. Thajeel, I. K., Samsudin, K., Hashim, S. J., & Hashim, F. (2023). Dynamic feature selection model for adaptive cross site scripting attack detection using developed multi-agent deep Q learning model. Journal of King Saud University-Computer and Information Sciences, 35(6), 101490.

6. Pardomuan, C. R., Kurniawan, A., Darus, M. Y., Mohd Ariffin, M. A., & Muliono, Y. (2023). Server-Side Cross-Site Scripting Detection Powered by HTML Semantic Parsing Inspired by XSS Auditor. Pertanika Journal of Science & Technology, 31(3).

7. Hu, T., Xu, C., Zhang, S., Tao, S., & Li, L. (2023). Cross-site scripting detection with two-channel feature fusion embedded in self-attention mechanism. Computers & Security, 124, 102990.

8. Kumar, A., & Sharma, I. (2023, April). Performance Evaluation of Machine

Learning Techniques for Detecting Cross-Site Scripting Attacks. In 2023 11th International Conference on Emerging Trends in Engineering & Technology-Signal and Information Processing (ICETET-SIP) (pp. 1-5). IEEE.

9. Panwar, P., Mishra, H., & Patidar, R. (2023). An Analysis of the Prevention and Detection of Cross Site Scripting Attack. International Journal, 11(1).

10. Lu, D., & Liu, L. (2023, February). Research on Cross-site Scripting Attack Detection Technology Based on Few-shot Learning. In 2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 6, pp. 1425-1429). IEEE.

11. Lee, S., Wi, S., & Son, S. (2022, April). Link: Black-box detection of cross-site scripting vulnerabilities using reinforcement learning. In Proceedings of the ACM Web Conference 2022 (pp. 743-754).

12. Chen, H. C., Nshimiyimana, A., Damarjati, C., & Chang, P. H. (2021, January). Detection and prevention of cross-site scripting attack with combined approaches. In 2021 International Conference on Electronics, Information, and Communication (ICEIC) (pp. 1-4). IEEE.

13. Rodríguez, G. E., Torres, J. G., Flores, P., & Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. Computer Networks, 166, 106960.

14. Gan, J. M., Ling, H. Y., & Leau, Y. B. (2021). A Review on detection of cross-site scripting attacks (XSS) in web security. In Advances in Cyber Security: Second International Conference, ACeS 2020, Penang, Malaysia, December 8-9, 2020, Revised Selected Papers 2 (pp. 685-709). Springer Singapore.

15. Usha, G., Kannimuthu, S., Mahendiran, P. D., Shanker, A. K., & Venugopal, D. (2020). Static analysis method for detecting cross site scripting vulnerabilities. International Journal of Information and Computer Security, 13(1), 32-47.

16. Sahoo, B., & Pattnaik, P. K. (2023). Cybersecurity Analysis and Phishing Attack. In Communication Technology and Gender Violence (pp. 31-37). Cham: Springer International Publishing.

17. Abd Alkareem, M. H. B., Nasif, F. Q., Ahmed, S. R., Miran, L. D., Algburi, S., & ALmashhadany, M. T. (2023, November). Linguistics for Crimes in the World by AI-Based Cyber Security. In 2023 7th International Symposium on Innovative Approaches in Smart Technologies (ISAS) (pp. 1-5). IEEE.

18. Baniya, D., & Chaudhary, A. (2023). Detecting cross-site scripting attacks using machine learning: A systematic review. Artificial Intelligence, Blockchain, Computing and Security Volume 1, 743-748.

19. Kirchner, R., Möller, J., Musch, M., Klein, D., Rieck, K., & Johns, M. Dancer in the Dark: Synthesizing and Evaluating Polyglots for Blind Cross-Site Scripting.

20. Khare, S., & Badholia, A. (2023). BLA2C2: Design of a novel blockchain-based light-weight authentication & access control layer for cloud deployments. International Journal on Recent and Innovation Trends in Computing and Communication, 11(3), 283-294.