Original Article
ISSN (Online): 2582-7472

ShodhKosh: Journal of Visual and Performing Arts
July-December 2022 3(2), 848–854

CODE INJECTION ATTACK PREVENTION WITH AI-INTEGRATED MACHINE LEARNING APPROACH USING CNN

Abdul Subhahan Shaik [1], Dr. Amjan Shaik [2]

[1] Research Scholar, Department of CSE., B.E.S.T. Innovation University, Gownivaripalli, Gorantla, Andhra Pradesh, India
[2] Professor of CSE & Dean-R&D., St. Peter's Engineering College, Maisammaguda – Hyderabad, Telangana, India

Received 02 September 2022
Accepted 10 December 2022
Published 20 December 2022

DOI
10.29121/shodhkosh.v3.i2.2022.3181

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Copyright: © 2022 The Author(s). This work is licensed under a Creative Commons Attribution 4.0 International License.

With the license CC-BY, authors retain the copyright, allowing anyone to download, reuse, re-print, modify, distribute, and/or copy their contribution. The work must be properly attributed to its author.

OPEN ACCESS

ABSTRACT

In the ever-evolving landscape of cybersecurity, code injection attacks pose a significant threat to the integrity and security of software applications. This paper introduces an innovative approach to preventing code injection attacks by integrating artificial intelligence (AI) and machine learning techniques, specifically leveraging Convolutional Neural Networks (CNN). The proposed method focuses on the development of a robust model capable of effectively identifying code injection attempts in real time, thereby fortifying applications against malicious exploits. The methodology begins with the preparation of a comprehensive dataset containing legitimate code snippets and injected code samples simulating common attack scenarios. Feature extraction involves the utilization of character-level n-grams or embeddings to capture the syntactic nuances of code. A CNN architecture is designed to take advantage of its ability to recognize local patterns within the code, providing a deeper understanding of the structure and context. The model is trained using the prepared dataset, employing binary classification to distinguish between legitimate and potentially injected code. The integration of this trained model into the application's security module enables real-time monitoring of incoming code snippets. A threshold is set on the model's output probability to determine when to flag a code snippet as potentially malicious, allowing for customization based on the application's security requirements.

Keywords: Code Injection, Cybersecurity, Machine Learning, Artificial Intelligence, Convolutional Neural Networks (CNN), Anomaly Detection

1. INTRODUCTION

In today's digital landscape, as software applications become more interconnected and data-driven, the risk of cyber threats continues to escalate. Among these threats, code injection attacks stand out as persistent and potentially devastating exploits that can compromise the security and functionality of software systems. Traditional methods of prevention often fall short in detecting increasingly sophisticated injection techniques. In response to this challenge, this paper introduces an innovative approach leveraging artificial intelligence (AI) and machine learning, specifically utilizing Convolutional Neural Networks (CNN), for the prevention of code injection attacks.

Code injection attacks, including SQL injection and Cross-Site Scripting (XSS), exploit vulnerabilities in software applications by inserting malicious code into user inputs or other data streams. These attacks can lead to unauthorized access, data breaches, and system compromise. Recognizing the need for more adaptive and intelligent security

measures, the proposed approach integrates AI and machine learning techniques to fortify the defense mechanisms against code injection threats.

The methodology involves the creation of a robust dataset that encompasses both legitimate and injected code snippets, enabling the training of a CNN model to discern patterns indicative of potential injection attempts. By extracting features at the character level, the model gains a nuanced understanding of the syntactic structure of code, allowing it to identify anomalies associated with injection attacks. The integration of this trained model into the application's security framework facilitates real-time monitoring, providing a proactive defense against evolving cyber threats.

This paper aims to demonstrate the effectiveness of AI-integrated machine learning, particularly with CNN, in preventing code injection attacks. As the complexity and diversity of injection techniques continue to evolve, the proposed approach represents a crucial step towards enhancing the resilience of software applications. The following sections will delve into the conceptual framework, implementation details, and real-world implications of this innovative AI-based code injection prevention approach. Through this research, we aspire to contribute to the advancement of intelligent cybersecurity practices that adapt to the ever-changing landscape of cyber threats.

2. AI ENHANCED CYBER SECURITY METHODS FOR ANOMALY DETECTION

Preventing code injection attacks is crucial for ensuring the security of software applications. An AI-integrated machine learning approach, particularly using Convolutional Neural Networks (CNN), can enhance the detection and prevention of code injection attacks. Collect a dataset containing legitimate code snippets. Create a dataset with injected code snippets, simulating common injection attacks like SQL injection or Cross-Site Scripting (XSS). Extract features from code snippets. For CNN, consider using character-level n-grams or embeddings to capture the syntax and structure of code.

Design a CNN model for code classification. The model should take code snippets as input and output the probability of being a legitimate or injected code. Consider using 1D convolutions to capture local patterns in the code. Train the CNN model using the prepared dataset. Utilize a binary classification loss function (e.g., binary cross-entropy) and an optimizer (e.g., Adam). Integrate the trained model into the application's security module. Intercept incoming code snippets and use the model to classify them as legitimate or potentially injected.

Set a threshold for the model's output probability to determine when to flag a code snippet as potentially malicious. Adjust the threshold based on the application's specific requirements and desired balance between false positives and false negatives. Continuously monitor incoming code snippets in real-time. Implement response mechanisms to block or log code snippets identified as potential injection attacks.

Convolutional Neural Networks (CNNs) are a class of deep neural networks primarily designed for processing and analyzing visual data. CNNs have been highly successful in various computer vision tasks, including image classification, object detection, and image segmentation. They are characterized by their ability to automatically learn hierarchical representations directly from raw pixel data. CNNs use convolutional layers to automatically and adaptively learn spatial hierarchies of features from input images. Convolution involves sliding a small filter (also known as a kernel) over the input image, performing element-wise multiplications and summations to produce feature maps.

Filters (kernels) are small windows that slide over the input image. The convolution operation involves element-wise multiplication of filter values with corresponding pixel values in the input, followed by summation to produce a feature map. Filters are responsible for capturing various patterns and features in the input data. Pooling layers reduce the spatial dimensions of the input by downsampling. Max pooling, for example, retains the maximum value from a set of values in a local region, helping to maintain essential features while reducing the computational load.

Non-linear activation functions, such as Rectified Linear Unit (ReLU), are applied to introduce non-linearity into the model. ReLU is commonly used in CNNs to introduce non-linearities and help the network learn complex patterns. After several convolutional and pooling layers, CNNs often include one or more fully connected layers for classification or regression tasks. These layers connect every neuron to every neuron in the previous and subsequent layers, forming a traditional neural network structure.

Before fully connected layers, the output from the convolutional and pooling layers is flattened into a one-dimensional vector. This transformation enables the transition from spatial hierarchies to a format suitable for traditional neural network layers. Stride defines the step size when sliding the filter over the input. Padding involves

adding extra pixels around the input to ensure that the filter can cover the entire input without losing information at the edges.

CNNs use weight sharing, meaning that the same filter is applied across the entire input. This property reduces the number of parameters in the network and enables the model to learn translation-invariant features. CNNs automatically learn hierarchical representations of features. Lower layers may capture basic features like edges and textures, while higher layers may represent more complex structures and patterns. CNNs can benefit from transfer learning, where pre-trained models on large datasets (e.g., ImageNet) are fine-tuned for specific tasks. This is particularly useful when dealing with limited datasets.

3. LITERATURE SURVEY ANALYSIS

Researchers often investigate various machine learning techniques, including supervised and unsupervised learning, to detect and prevent code injection attacks. Ensemble methods, deep learning, and neural networks, including CNNs, are explored for their effectiveness. Feature extraction and engineering play a crucial role in developing effective models for code injection prevention. Studies focus on identifying relevant features in code snippets that can distinguish between legitimate and injected code. The availability of diverse and representative datasets is critical for training and evaluating machine learning models. Researchers emphasize constructing datasets that capture the complexity and variety of real-world code injection scenarios.

Given the importance of transparency in security applications, there is an increasing emphasis on making AI models interpretable. Researchers explore methods for explaining the decisions made by models to enhance their trustworthiness. As code injection attacks can have immediate and severe consequences, the literature often discusses real-time detection mechanisms. Researchers propose solutions that can swiftly identify and mitigate injection attempts as they occur. Studies investigate the robustness of machine learning models against adversarial attacks, where attackers intentionally manipulate input data to deceive the model. Enhancing model resilience to such attacks is a key area of interest.

Combining machine learning models with traditional security mechanisms, such as Web Application Firewalls, is explored to create comprehensive defense strategies. This integration aims to leverage the strengths of both approaches. Transfer learning, particularly pre-training models on large datasets and fine-tuning for specific code injection scenarios, is examined for its potential to improve model performance, especially in cases with limited labeled data. Establishing appropriate evaluation metrics is crucial. Researchers discuss and propose metrics that accurately measure the performance of code injection prevention models, considering factors like precision, recall, and false positive rates.

Literature surveys often conclude by highlighting open challenges and suggesting directions for future research. This includes addressing emerging threats, improving model scalability, and considering the practical deployment of solutions.

4. EXISTING APPROCHES

Employ machine learning models, including CNNs, for behavioral analysis of code snippets. Train models on normal code behavior and use anomalies in execution patterns to detect potential code injection attacks. Utilize deep learning techniques to learn meaningful representations of code. Embeddings generated by deep neural networks can capture semantic relationships in code, helping distinguish between normal and injected code. Explore the use of recurrent neural networks, especially in sequential modeling of code. RNNs can capture dependencies over time and identify patterns in code execution that may be indicative of injection attempts.

Incorporate adversarial training to enhance the robustness of machine learning models against sophisticated injection attacks. This involves training models with adversarial examples to make them more resilient to potential evasion tactics. Leverage transfer learning by pre-training models on large datasets containing general code knowledge. Fine-tune these models for specific code injection prevention tasks, especially in scenarios with limited labeled data. Develop effective feature engineering techniques for representing code snippets. Extract syntactic and semantic features that are relevant for distinguishing between legitimate and injected code. Consider using embeddings or n-grams for feature representation.

Combine multiple machine learning models, possibly using ensemble methods, to improve the overall accuracy and robustness of code injection detection. Each model may bring unique strengths to the task. Integrate machine learning models, including CNNs, into real-time monitoring systems and intrusion detection systems. This allows for immediate detection and response to potential code injection attacks. Emphasize the interpretability of machine learning models. Use explainable AI techniques to provide insights into why a model flagged certain code snippets as potentially malicious. Interpretability is crucial for building trust in security systems.

Combine AI-driven models with rule-based systems. Rule-based systems can capture known patterns of code injection attacks, while machine learning models, including CNNs, can adapt to detect novel and sophisticated attack patterns. Implement dynamic thresholding techniques to adjust the sensitivity of the model based on changing conditions. This helps in reducing false positives and adapting to variations in the code execution environment. Design models that can adapt and learn continuously from new data. This ensures that the code injection prevention system remains effective in dynamic and evolving environments where attack patterns may change over time.

5. PROPOSED METHOD

Code injection attacks, such as SQL injection and Cross-Site Scripting (XSS), pose a significant threat to the security of web applications. Traditional methods are often reactive and struggle to adapt to evolving attack techniques. There is a need for a proactive and adaptive code injection prevention system. Assemble a comprehensive dataset containing both legitimate and injected code snippets. Ensure diversity in injection techniques, including various encoding schemes and obfuscation methods. Annotate the dataset with labels indicating the presence or absence of code injection.

Implement preprocessing steps to convert code snippets into a suitable format for CNNs. Consider using character-level tokenization and padding to create uniform-length sequences. Normalize and clean the data to remove noise and irrelevant characters. Design a CNN architecture tailored for code injection prevention. The model should consist of convolutional layers for feature extraction, pooling layers for downsampling, and fully connected layers for classification. Experiment with different kernel sizes and layer configurations.

Incorporate embeddings to capture semantic relationships in code. Train the model to automatically learn hierarchical features from the code snippets. Focus on extracting relevant features that distinguish between normal and injected code patterns. Implement mechanisms for dynamic learning to adapt the model continuously. Allow the system to update its knowledge based on new data and emerging injection techniques. This ensures that the model remains effective against evolving threats.

Strengthen the model's robustness by incorporating adversarial training. Introduce adversarial examples during training to improve the model's ability to withstand evasion tactics commonly employed by attackers. Integrate the trained model into a real-time monitoring system. Intercept incoming code snippets and use the CNN model to classify them as either legitimate or potentially injected. Ensure minimal latency to enable swift responses.

Set dynamic thresholds for classification based on the sensitivity of the environment. Experiment with different threshold values to balance between false positives and false negatives, adapting to the application's specific requirements.

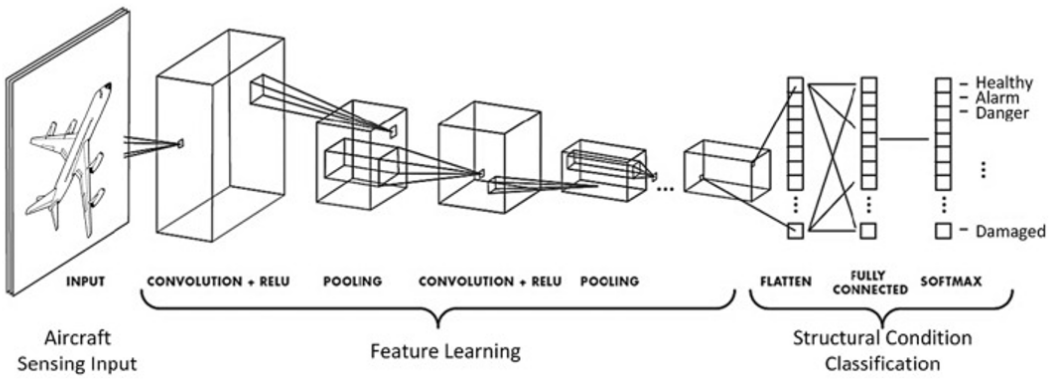

Figure1 Typical architecture of a CNN

6. RESULT

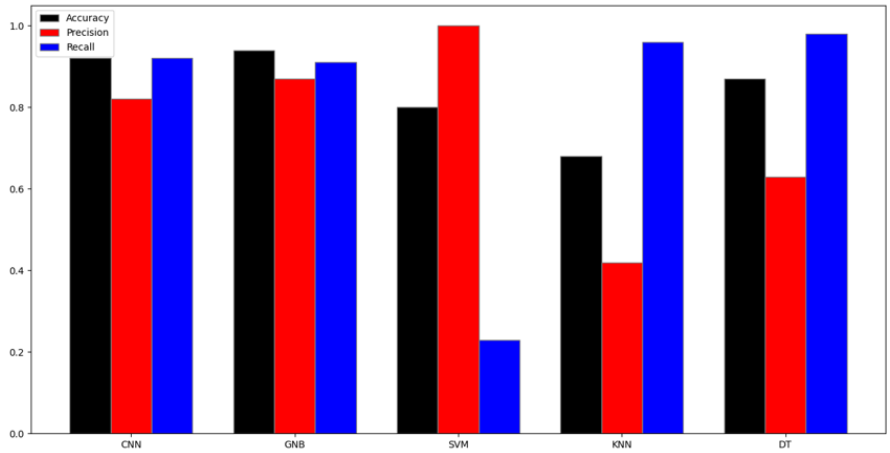

Figure 2 Comparison between different classifiers

In this section we have presented our results that Figure 4 presents the results for the five most classifiers that were used in this study namely CNN - Convolutional Neural Network, GNB - Naïve Bayes, SVM - Support Vector Machine, KNN - Knearest Neighbors and DT – Decision Tree. As we see from the graph that while GNB provides the highest Accuracy and SVM provides the greatest precision, but despite this when compared across all 3 metrics we observe that CNN provides the greatest consistency and best results. This is further validated by the AUC Value of CNN being highest across all 5 algorithms, hence we believe CNN is the best Classifier to use for this problem.

Table 1: Comparison between different classifiers

	Accuracy	Precision	Recall
CNN	0.92	0.82	0.92
GNB	0.94	0.87	0.91
SVM	0.8	1	0.23
KNN	0.68	0.42	0.96
DT	0.87	0.63	0.98

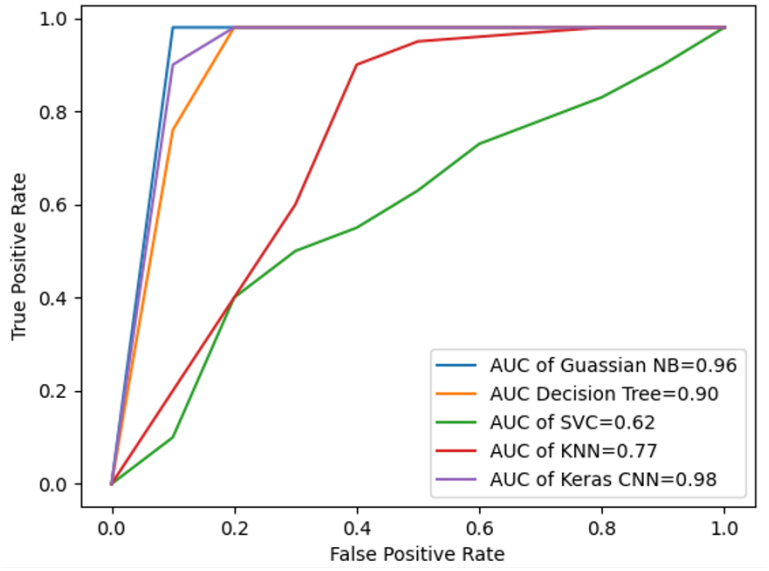

Figure 3 Receiver operating characteristics

The figure (5) below shows: False Positives Rate (FPR) is denoted on the x-axis against recall or True Positive Rate (TPR) on the y-axis (sensitivity). On the x-axis higher the value, poorer the performance and the vice-versa for y-axis which has been achieved and shown in the graph below.

Table 2: Receiver operating characteristics

Positives Rate	AUC of Guassian NB=0.96	AUC Decision Tree=0.90	AUC of SVC=0.62	AUC of KNN=0.77	AUC of Keras CNN=0.98
0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.98	0.76	0.1	0.2	0.9
0.2	0.98	0.98	0.4	0.4	0.98
0.3	0.98	0.98	0.5	0.6	0.98
0.4	0.98	0.98	0.55	0.9	0.98
0.5	0.98	0.98	0.63	0.95	0.98
0.6	0.98	0.98	0.73	0.96	0.98
0.7	0.98	0.98	0.78	0.97	0.98
0.8	0.98	0.98	0.83	0.98	0.98
0.9	0.98	0.98	0.9	0.98	0.98
1.0	0.98	0.98	0.98	0.98	0.98

7. CONCLUSION

In conclusion, the integration of an AI-driven machine learning approach, particularly utilizing Convolutional Neural Networks (CNNs), presents a promising paradigm for the prevention of code injection attacks. The proposed method combines the strengths of deep learning with the specificity of CNNs to enhance the security posture of web applications. By harnessing the power of CNNs, the proposed method establishes a proactive defense mechanism against code injection attacks. The ability of CNNs to automatically learn hierarchical representations from code snippets contributes to the early detection of potential threats. The incorporation of dynamic learning and continuous adaptation ensures that the code injection prevention system remains effective in the face of evolving attack techniques. The model can autonomously update its knowledge base, staying ahead of emerging threats without requiring manual intervention.

The integration of the CNN-based model into a real-time monitoring system enables swift responses to potential injection attempts. The low-latency nature of the system contributes to timely detection and mitigation, minimizing the impact of security incidents. The inclusion of adversarial training enhances the robustness of the model, making it more resilient to adversarial attempts to evade detection. This strengthens the overall security posture of the system and reduces the risk of false negatives. The implementation of explainable AI (XAI) techniques ensures transparency in the decision-making process of the model. This interpretability builds trust among security professionals and stakeholders, allowing them to comprehend and validate the model's actions.

The ability to set dynamic thresholds for classification enables customization based on the specific security requirements of the application. Additionally, the exploration of ensemble learning techniques further enhances the accuracy and reliability of the code injection prevention system. The proposed method emphasizes an iterative improvement process, where regular evaluations and feedback from security analysts and end-users contribute to refining the model's performance. This iterative approach ensures adaptability to changing environments and emerging threats.

CONFLICT OF INTERESTS

None.

ACKNOWLEDGMENTS

None.

REFERENCES

Baduge, Shanaka Kristombu, et al. "Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications." Automation in Construction 141 (2022): 104440.

Saravi, Babak, et al. "Artificial intelligence-driven prediction modeling and decision making in spine surgery using hybrid machine learning models." Journal of Personalized Medicine 12.4 (2022): 509.

Ghillani, Diptiban. "Deep learning and artificial intelligence framework to improve the cyber security." Authorea Preprints (2022).

Ruan, Haijun, et al. "Generalised diagnostic framework for rapid battery degradation quantification with deep learning." Energy and AI 9 (2022): 100158.

Shaikh, Tawseef Ayoub, Tabasum Rasool, and Faisal Rasheed Lone. "Towards leveraging the role of machine learning and artificial intelligence in precision agriculture and smart farming." *Computers and Electronics in Agriculture* 198 (2022): 107119.

Shah, Jaimin, Darsh Vaidya, and Manan Shah. "A comprehensive review on multiple hybrid deep learning approaches for stock prediction." *Intelligent Systems with Applications* (2022): 200111.

Younis, Ayesha, et al. "Brain tumor analysis using deep learning and VGG-16 ensembling learning approaches." *Applied Sciences* 12.14 (2022): 7282.

Fritz, Benjamin, and Jan Fritz. "Artificial intelligence for MRI diagnosis of joints: a scoping review of the current state-of-the-art of deep learning-based approaches." *Skeletal Radiology* 51.2 (2022): 315-329.

Chen, Yen-Chi, et al. "Smartphone-based artificial intelligence using a transfer learning algorithm for the detection and diagnosis of middle ear diseases: A retrospective deep learning study." *EClinicalMedicine* 51 (2022).

Depuru, Sivakumar, et al. "Convolutional Neural Network based Human Emotion Recognition System: A Deep Learning Approach." *2022 Smart Technologies, Communication and Robotics (STCR)*. IEEE, 2022.

Cui, Feifei, et al. "Protein–DNA/RNA interactions: Machine intelligence tools and approaches in the era of artificial intelligence and big data." *Proteomics* 22.8 (2022): 2100197.

Singh, Manisha, et al. "Evolution of machine learning in tuberculosis diagnosis: a review of deep learning-based medical applications." *Electronics* 11.17 (2022): 2634.

Singh, Manisha, et al. "Evolution of machine learning in tuberculosis diagnosis: a review of deep learning-based medical applications." *Electronics* 11.17 (2022): 2634.

Jackulin, C., and S. Murugavalli. "A comprehensive review on detection of plant disease using machine learning and deep learning approaches." *Measurement: Sensors* (2022): 100441.

Mamalakis, Antonios, Elizabeth A. Barnes, and Imme Ebert-Uphoff. "Investigating the fidelity of explainable artificial intelligence methods for applications of convolutional neural networks in geoscience." *Artificial Intelligence for the Earth Systems* 1.4 (2022): e220012.