GenXSS: an AI-Driven Framework for Automated Detection of XSS Attacks in WAFs

Vahid Babaey * and Arun Ravindran

Posted Date: 5 March 2025

doi: 10.20944/preprints202503.0313.v1

Keywords: XSS; generative AI; LLM; WAF; ModSecurity WAF; AWS WAF; cybersecurity

Article

GenXSS: An AI-Driven Framework for Automated Detection of XSS Attacks in WAFs

Vahid Babaey [1,*] and Arun Ravindran [2]

[1] Department of Electrical and Computer Engineering, University of North Carolina at Charlotte, Charlotte, NC 28223, USA
[2] Department of Electrical and Computer Engineering, University of North Carolina at Charlotte, arun.ravindran@charlotte.edu
* Correspondence: vbabaey@charlotte.edu

Abstract: The increasing reliance on web services has led to a rise in cybersecurity threats, particularly Cross-Site Scripting (XSS) attacks, which target client-side layers of web applications by injecting malicious scripts. Traditional Web Application Firewalls (WAFs) struggle to detect highly obfuscated and complex attacks, as their rules require manual updates. This paper presents a novel generative AI framework that leverages Large Language Models (LLMs) to enhance XSS mitigation. The framework achieves two primary objectives: (1) generating sophisticated and syntactically validated XSS payloads using in-context learning, and (2) automating defense mechanisms by testing these attacks against a vulnerable application secured by a WAF, classifying bypassing attacks, and generating effective WAF security rules. Experimental results using GPT-4o demonstrate the framework's effectiveness generating 264 XSS payloads, 83% of which were validated, with 80% bypassing ModSecurity WAF equipped with an industry standard security rule set developed by the Open Web Application Security Project (OWASP) to protect against web vulnerabilities. Through rule generation, 86% of previously successful attacks were blocked using only 15 new rules. In comparison, Google Gemini Pro achieved a lower bypass rate of 63%, highlighting performance differences across LLMs.

Keywords: XSS; generative AI; LLM; WAF; ModSecurity WAF; AWS WAF; cybersecurity

1. Introduction

The rise of web services has significantly influenced how organizations manage client information. However, as online platforms expand, so do the threats to user data security, with Cross-Site Scripting (XSS) emerging as a prevalent and dangerous attack vector [1]. XSS attacks exploit website vulnerabilities to inject malicious scripts, often through links, URLs, or user inputs, which execute within the victim's browser. Web Application Firewalls (WAFs) serve as a primary defense against XSS by detecting and blocking malicious payloads [2]. However, as web attacks grow more sophisticated, traditional WAFs often struggle to adapt to advanced and obfuscated patterns [3].

Generative AI has emerged as a promising solution, leveraging Large Language Models (LLMs) to generate text, code, and images [4]. LLMs, when combined with techniques like in-context learning and fine-tuning, excel at generating new data from a few relevant examples. [5,6]. This adaptability makes LLMs particularly effective for enhancing WAF defenses by generating simulated attack scenarios based on real-world examples [7]. These generated samples can then be analyzed and tested to improve the detection capabilities of WAFs, ultimately contributing to the development of more robust cybersecurity defenses.

In this paper, we propose **GenXSS**, a generative AI framework for enhancing WAF defenses against XSS attacks. Our framework achieves two primary objectives:

1. **Generation of Obfuscated XSS Attacks:** Using LLMs and curated in-context learning, GenXSS generates complex XSS payloads validated against real-world vulnerable applications.
2. **Automated Defense Mechanisms:** The framework identifies bypassing payloads and employs machine learning and LLMs to generate and validate new WAF rules.

Experimental results demonstrate the framework's efficacy with 264 new XSS payloads generated, 83% were syntactically validated using a buggy web application, ensuring they were correctly structured and executable. Among these validated attacks, 80% bypassed ModSecurity WAF equipped with the latest OWASP Core Rule Set [8], and nearly 100% bypassed AWS WAF. By classifying the generated payloads using our machine-learning algorithms and generating rules through the GPT-4o LLM rule-generation methodology, we developed and validated security rules that effectively blocked 86% of all previously successful attacks which bypassed ModSecurity with only 15 new rules. We also employed Gemini Pro LLM to generate new XSS attacks, resulting in a total of 220 generated samples. Of these, 140 (63%) were validated as syntactically correct and effective, with 104 (74%) successfully bypassing ModSecurity.

The contributions of this paper include an innovative generative AI framework for XSS mitigation including in-context learning involves incorporating task-specific examples directly within the input prompt to guide the model's behavior and output generation, automated WAF rule optimization, and experimental validation on real-world WAFs. The remainder of the paper is structured as follows: Section 2 provides a background on XSS attacks and generative AI. Section 3 reviews related literature. Section 4 details the framework. Section 5 presents results, followed by discussion in Section 6, and conclusions in Section 7.

2. Background

2.1. XSS Attacks

Cross-site Scripting attacks (XSS) can be used by attackers to undermine application security in many ways. XSS vulnerabilities have been used to create social networks worms, steal cookies, spread malware, deface websites, and phish for credentials [1].

Cross-Site Scripting (XSS) attacks come in three main types: Reflected, DOM-based and Stored. Reflected XSS (Non-persistent XSS), the most common type, involves the attacker embedding their payload in a request sent to the web server. The server reflects the payload back in its response, executing it in the victim's browser. DOM-based XSS is a more advanced, client-side attack where the malicious script exploits vulnerabilities in the web application's client-side scripts to manipulate the Document Object Model (DOM), executing the payload within the browser. This type of XSS targets elements like URLs or referrers directly in the DOM. Stored XSS (Persistent XSS) occurs when an attacker embeds malicious scripts, such as JavaScript, into a target application. These scripts are typically injected through user input fields, such as comments or posts, and are stored on the server, allowing them to execute whenever other users access the affected content.[1].

2.2. Web Application Firewalls (WAFs)

A Web Application Firewall (WAF) is a security mechanism that protects web applications by monitoring and blocking malicious HTTP/S traffic. WAFs typically use two strategies: **rule-based** and **machine-learning-based** approaches. Rule-based methods rely on predefined attack signatures using regular expressions, offering transparency, low false positive rates, and quick deployment, but they are limited to known patterns, require frequent updates, and struggle with scalability. Machine-learning-based methods, on the other hand, detect attacks by learning from data, making them adaptable to novel or obfuscated patterns and capable of detecting anomalies beyond static signatures. However, these methods can have high false positive rates and depend heavily on high-quality labeled data for training and fine-tuning [2].

2.3. Generative AI

Generative AI is a field of artificial intelligence focused on creating new content, such as text, images, audio, or code, by learning patterns from existing data [5]. Central to this field are Large Language Models (LLMs), such as GPT (Generative Pre-trained Transformer), which use deep-learning architectures to generate coherent and contextually relevant language. These models leverage transformer architectures to capture complex relationships within data and utilize multi-layered neural networks to process vast datasets by extracting increasingly abstract features. With billions or trillions of parameters, LLMs are highly complex and demand substantial computational resources for training.

3. Related Work

While LLMs have shown promise in cybersecurity applications, there are currently limited documented real-world case studies explicitly detailing their deployment in active cybersecurity environments. Most implementations remain in research and experimental phases, focusing on theoretical frameworks or controlled test environments. In this section, we review related work focusing on XSS attack generation, rule adaptation methodologies, and approaches for enhancing WAF robustness.

Wu et al. [9] proposed an RNN-based generator for creating malicious payloads, including XSS and SQL injection, validated by a payload corrector. These payloads were tested against a shadow model mimicking WAF behavior and then real-world WAFs. Successfully bypassing payloads were used to enhance WAF detection by updating rules with new signatures. Yao et al. [10] developed a method for generating XSS attack vectors using an improved Dueling Deep Q-Network (DDQN) algorithm modeled as a Markov Decision Process. Priority experience replay and a reward function based on edit distance guided effective mutations. Generated vectors were validated through semantic analysis and tested against WAFs in both proxy and direct connection modes. Garn et al. [11] employed combinatorial testing with an attack grammar to create diverse XSS payloads, defined by attributes, payloads, and tags. The payloads were tested against multiple WAFs in controlled environments, and their effectiveness was compared to state-of-the-art static attack lists. Alaoui et al. [12] used a Generative Adversarial Network (GAN) to create adversarial XSS attacks by modifying existing dataset samples to evade detection by an LSTM-based XSS detection model. The crafted attacks significantly reduced the detection model's performance, highlighting their effectiveness. Khan [13] introduced a generative AI model combining auto-regressive and transformer techniques to generate XSS payloads by analyzing backend and frontend code. The generated payloads were tested for effectiveness using the OWASP Juice Shop application.

Our approach differs from traditional methods, where security experts manually update WAF rules in response to observed threats. Furthermore, the works described above present challenges for anomaly detection models that use neural networks. These challenges arise due to the substantial training data required and the complexity of neural network design and implementation. Additionally, we observe that existing XSS attack generation methods in the literature treat attack generation and mitigation as separate processes. In contrast, our framework integrates generative AI to simultaneously automate the generation of XSS attacks and the creation of corresponding WAF rules, offering a unified and adaptive defense mechanism.

4. GenXSS Framework

In this section, we present the GenXSS framework, a general and flexible solution designed to enhance the performance of WAFs against XSS attacks. The primary advantage of the GenXSS framework lies in its ability to both generate and defend against XSS attacks while offering adaptability to various parameters, including the LLM model, application, and WAF. The framework can be configured to suit different environments based on specific requirements.

4.1. Architecture

In the proposed framework, we utilize in-context learning by providing a set of valid, carefully crafted attacks as examples within the prompt to guide the LLM in generating new XSS samples. However, since LLMs can occasionally produce irrelevant or inaccurate samples, the generated outputs must be validated against a vulnerable application specifically designed to be susceptible to XSS attacks. This validation ensures that only effective and legitimate samples are retained, while irrelevant samples are discarded. Figure 1 illustrates the architecture of GenXSS.

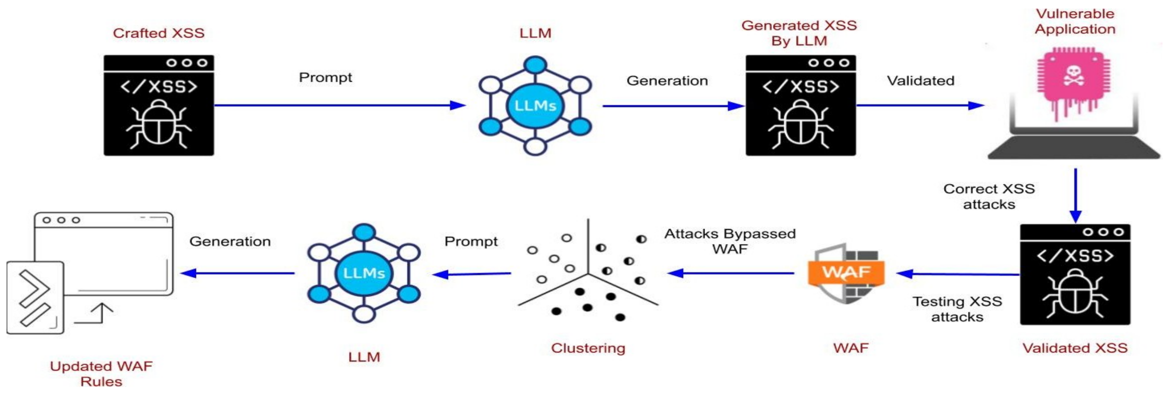

Figure 1. GenXSS framework architecture.

The next step involves testing the validated samples using a Web Application Firewall (WAF). Samples that successfully bypass the WAF are stored separately and then processed using machine-learning clustering methods. This clustering is a crucial step which group similar attack types for the subsequent generation of security rules to update the WAF. In the second stage of the framework, we use another LLM to generate effective and comprehensive WAF rules tailored to the characteristics of the identified clusters. To maximize rule effectiveness, we apply a Reinforcement Learning with Human Feedback (RLHF) approach. This iterative refinement process ensures the successful mitigation of all XSS attacks.

In the remainder of this section, we describe each of these steps in detail.

4.1.1. **XSS Generation**

In our written prompt, the in-context learning examples were manually crafted and validated against a vulnerable application to ensure they represent real attacks capable of bypassing the WAF. These examples serve as a foundation for the LLM to generate obfuscated XSS attacks. The examples are designed to be complex, clear, and effective to guide the LLM in producing more obfuscated attack samples. The construction of the prompt includes (1) **Problem description:** A clear explanation of the model's purpose and goal, along with examples provided using few-shot learning. (2) **Instructions for In-Context Learning:** Systematic obfuscation techniques that guide the LLM to generate new and more obfuscated attacks. (3) **Tasks:** Precise and detailed tasks that specify exactly what needs to be generated, ensuring the outputs are diverse, comprehensive, and suitable for real-world testing.

4.1.2. **XSS Validation**

The validation process was conducted using Brute Logic [14], a platform specifically designed for testing XSS attacks. XSS examples used for in-context learning were first validated in this application to ensure their accuracy. Brute Logic focuses exclusively on XSS vulnerabilities, providing realistic scenarios such as comment sections, search bars, and URL parameters. It also supports various XSS contexts, including JavaScript event handlers, HTML attributes, and DOM-based injections, making it a versatile tool for comprehensive XSS testing. Additionally, the XSS examples were validated against a WAF to evaluate their ability to bypass existing defenses. Similarly, LLM-generated XSS attacks were first tested with Brute Logic to ensure syntactic correctness and subsequently with a WAF to verify their ability to exploit vulnerabilities.

4.1.3. **XSS Clustering**

To cluster and analyze the generated XSS attacks, we utilized two machine-learning algorithms: TF-IDF with Hierarchical Agglomerative Clustering (HAC) and SequenceMatcher with Density Based Spatial Clustering of Applications with Noise (DBSCAN). **TF-IDF + HAC:** The primary strategy of this algorithm is to emphasize unique terms while de-emphasizing common terms. Queries are represented as vectors using TF-IDF, and HAC clusters them based on their similarity [15]. For example, -alert%0d%0a/**//*test*/(1)- and -alert%0a/**//*test*/(1)- are grouped due to their shared terms. **SequenceMatcher + DBSCAN:** This algorithm calculates character-level similarity scores between queries using SequenceMatcher. DBSCAN then clusters queries with high similarity density [16]. For instance, -alert(1)%0d%0a//%20comment- and -alert(1)//comment%0d%0a- are grouped due to their structural similarity. The integration of TF-IDF with HAC and SequenceMatcher with DBSCAN provides a robust solution for clustering XSS payloads by leveraging their complementary strengths. TF-IDF + HAC effectively extracts features and captures hierarchical relationships without predefined cluster shapes, while SequenceMatcher + DBSCAN excels in identifying string-level similarities, managing overlapping clusters, and dynamically determining cluster counts. These approaches outperform traditional methods like k-means and regular expression-based clustering.

4.1.4. **WAF Security Rule Generation**

The features of each cluster, along with some of the generated XSS attacks, can be utilized to create a comprehensive and structured prompt for the LLM to generate new security rules for updating WAF security rules. The prompt includes (1) **System Role:** Assign the LLM the role of a security expert to ensure the generation of efficient security rules specifically designed to block XSS attacks. (2) **Cluster Characteristics:** Provide a detailed description of each cluster along with a few representative examples. This enables the LLM to analyze the features of each cluster and craft security rules tailored to their specific characteristics. (3) **Task Definition:** Define the task explicitly to ensure the generated rules are syntactically correct, achieve a high true positive rate, and minimize false positives. The rules are presented as a continuous block for seamless integration, with comments included to explain each rule's purpose and effectiveness.

4.2. Role of Reinforcement Learning with Human Feedback

One major challenge in using LLMs is generating accurate outputs, often hindered by unstructured prompts leading to syntax errors and incomplete coverage. Reinforcement Learning with Human Feedback (RLHF) addresses this by iteratively refining prompts based on feedback about issues like redundancy and errors, ensuring improved results over time. Specifically in the GenXSS framework, RLHF enhances rule generation by maximizing true positive rates and minimizing false negatives. and ensuring syntactically correct outputs.

5. Evaluation and Results

This section evaluates GenXSS based on three aspects: its ability to bypass WAFs, the effectiveness of generated security rules in mitigating attacks, and the adaptability of the methodology to different LLMs.

5.1. Experimental Setup

Our experimental setup utilized OpenAI GPT-4o and Google Gemini Pro as the Large Language Models (LLMs) for attack generation. The system operated on Ubuntu 22.04 LTS with Apache v2.4.52 as the web server. For web security, ModSecurity v2.9.5 and AWS WAF were employed as Web Application Firewalls (WAFs), with attack detection rules based on OWASP CRS v4.9.0. The Brute Logic application was used as the vulnerable testing environment, and clustering algorithms are TF-IDF + HAC and DBSCAN + SequenceMatcher for data analysis. GPT-4o was configured with a temperature setting of 0.7, and Gemini Pro with a temperature setting of 1. Since our primary goal was to generate diverse yet functional XSS payloads, we relied on these preset values rather than manually optimizing them. While higher temperature typically increases randomness, potentially leading to more invalid payloads, our study did not explicitly measure its effect on validity.

5.2. Sample XSS Generation

To illustrate how GenXSS operates, we present two examples of generated XSS attacks: one that is relevant and successfully bypasses the ModSecurity WAF, and another that is irrelevant.

5.2.1. **\";\u0061\u006c\u0065\u0072\u0074(1);//**

This attack which is a relevant one begins with an escape character, a closing double-quote ("), followed by a semicolon (;). This sequence is intended to terminate any existing JavaScript or HTML attribute context prematurely. This is a common technique to escape out of the current context. Next, the query includes Unicode encoding (\uXXXX), which represents characters in hexadecimal notation. For instance, \u0061\u006c\u0065\u0072\u0074 decodes to the JavaScript function `alert`.

The payload of the attack is the function call `alert(1)`, which, when executed, displays an alert box with the value `1`.

Finally, the query includes a comment (//), which indicates the start of a single-line comment in JavaScript. Anything following it on the same line is ignored. This neutralizes any remaining code on the line that could interfere with the attack. The payload is injected using a URL-based attack: https://brutelogic.com.br/gym.php?p16=red\%22;\u0061 \u006c\u0065\u0072\u0074(1);//

5.2.2. **\";\u0061l\x65rt(1);//**

This payload which is an irrelevant sample fails due to mixed encoding styles \u0061l \x65rt(1), making the “alert" function unrecognizable by the JavaScript parser. The inconsistency prevents the script from executing, rendering the payload ineffective.

5.2.3. **Bypassing ModSecurity**

The first payload bypasses ModSecurity by employing several advanced techniques. First, it utilizes Unicode obfuscation, where Unicode escape sequences (\uXXXX) represent characters in the alert function (\u0061\u006c\u0065\u0072\u0074), effectively evading detection. Second, context breaking is achieved using the \"; sequence, which closes an existing string or attribute context and introduces a semicolon (;) to initiate a new statement, disrupting normal parsing mechanisms. Additionally, the payload leverages single-line comments (//) to neutralize any trailing code that might interfere with execution. Finally, it avoids the use of suspicious characters commonly flagged by WAFs, such as <, >, or parentheses (()), further reducing its detectability.

5.2.4. **Results**

Table 1 summarizes the results of XSS generation using the GPT-4o and Gemini Pro LLM models. Both models were provided with the same prompts along with 4 manually crafted obfuscated XSS samples as in-context examples. The generated samples were validated for correctness using the Brute Logic web application.

Table 1. XSS attack generation results for GPT-4o and Gemini-Pro.

LLM Model	Num XSS Attacks	Num Valid	Num Invalid
GPT-4o	264	220	44
Gemini-Pro	220	140	80

- GPT-4o: Generated 264 samples, of which 220 were valid XSS payloads, achieving a validity rate of 83%.
- Gemini: Generated 220 samples, of which 140 were valid, resulting in a validity rate of 63%.

Tables 2 and 3 provide a breakdown of the performance of ModSecurity WAF against the XSS attacks generated by GPT-4o and Gemini-Pro respectively. The XSS attacks are listed by type of attack. The generated samples were clustered by type, as each type utilizes a distinct prompt and exhibited unique structures and patterns. The dataset imbalance stems from real-world attack feasibility, as Reflected XSS had four successful manually crafted payloads, while DOM-based XSS had only one, and Stored XSS could not be included due to the lack of a bypassing example.

Table 2. Validated XSS attacks by type generated by GPT-4o that bypassed ModSecurity.

Attack Types	Num XSS Attacks	Num Blocked	Num Bypass WAF
Reflected	178	34	144
Dom-Based	42	12	30

Table 3. Validated XSS attacks by type generated by Gemini Pro that bypassed ModSecurity.

Attack Types	Num XSS Attacks	Num Blocked	Num Bypass WAF
Reflected	116	22	94
Dom-Based	24	14	10

- GPT-4o: Of the 220 validated XSS attacks, 174 (80%) bypassed ModSecurity.
- Gemini Pro: Of the 140 validated XSS attacks, 104 (74%) bypassed ModSecurity

Table 4 presents the results of GPT-4o generated XSS attacks tested against AWS WAF configured with the Core Rule Set and attached to an Application Load Balancer (ALB). Results indicate that all but one generated attack successfully bypassed the AWS WAF.

Table 4. GPT-4o XSS attacks validation that bypassed AWS WAF.

Attack Types	Num XSS Attacks	Num Blocked	Num Bypass WAF
Reflected	178	1	177
Dom-Based	42	0	42

Table 5 presents the evaluation results for ModSecurity rules generated using the two clustering techniques described in Section 4. For TF-IDF + HAC, we utilized Ward's linkage metric, which minimizes the variance within clusters, with a distance threshold of 1.8. This approach was chosen to identify a meaningful number of clusters without requiring a predefined cluster count. For DBSCAN, the metric parameter was set using a custom distance matrix, precomputed with a similarity ratio of 1 from SequenceMatcher. The parameters `eps = 0.1` and `min_samples = 2` were selected after testing, as they produced a minimal yet meaningful set of clusters. To evaluate the cohesion and separation of the clusters, silhouette scores were used, which range from -1 to $+1$, with $+1$ indicating strong clustering and -1 indicating poor clustering. The silhouette scores for TF-IDF + HAC and SequenceMatcher + DBSCAN were 0.18 and 0.32, respectively, reflecting moderate clustering performance.

Table 5. Clustering Results for XSS Attack Types

Attack Types	Num Samples	Clustering Algorithms	Num Rules	Num Blocked
Reflected	144	TF-IDF + HAC	9	120
		SeqMatcher + DBSCAN	5	114
Dom-Based	30	TF-IDF + HAC	6	30
		SeqMatcher + DBSCAN	4	16

The generated clusters were utilized by the LLM for rule generation, with TF-IDF + HAC clustering showing the best performance. Using this approach, GPT-4o generated 15 SecRules that successfully blocked 83% of the 144 XSS attacks that previously bypassed ModSecurity. Table 6 presents metrics such as False Positive, Precision, Recall, and F1 Score to evaluate the rules' effectiveness in detecting true positives while minimizing errors. For this study, we used a dataset with a 4:1 ratio of normal to attack samples (800 normal and 220 attack samples).

Table 6. Performance metrics for generated ModSecurity security rules.

Metric	**Definition/Calculation**
True Positives (TP)	Number of attacks correctly blocked by the WAF: $TP = 150$
False Negatives (FN)	Number of attacks not blocked by the WAF: $FN = 24$
True Negatives (TN)	Number of normal samples correctly not blocked by the WAF: $TN = 800$
False Positives (FP)	Number of normal samples incorrectly blocked by the WAF: $FP = 0$
Accuracy	$\frac{\text{TN+TP}}{\text{TN+TP+FN+FP}} = 0.9753$
Precision	$\frac{\text{TP}}{\text{TP+FP}} = 1.0$
Recall	$\frac{\text{TP}}{\text{TP+FN}} = 0.8621$
F1-Score	$2 \cdot \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision+Recall}} = 0.9259$

6. Discussion

In this paper, we presented GenXSS, a generative AI-based framework designed to combat highly obfuscated XSS attacks by synthesizing novel attack patterns capable of bypassing Web Application Firewall (WAF) security rules. Unlike traditional manual methods, which require extensive expertise and struggle to address novel patterns, GenXSS integrates machine learning and generative AI to efficiently create scalable defense rules for entire classes of attacks. While generative AI enhances scalability, it also introduces challenges such as computational costs which affect scalability in real-world deployment, token limitations affecting attack diversity, and occasional inaccuracies due to the probabilistic nature of LLMs. Additionally, some publicly available models, such as Anthropic Claude and Meta's LLaMA, are restricted from generating XSS attack vectors. While these limitations are intended to prevent misuse by attackers, they also hinder defenders from utilizing these models to develop proactive defense mechanisms. The framework's validation was limited to two types of XSS attacks, a single vulnerable application, and two WAFs ModSecurity and AWS WAF. We also conducted preliminary testing on Cloudflare and Imperva WAFs. These WAFs require domain registration, introducing complexity in controlled experimentation. Also the recall score of 0.86 reflects that while our WAF rule-generation process significantly improved detection, some attacks still bypassed due to limitations in the clustering-based approach. Despite these limitations, GenXSS demonstrates significant potential to enhance cybersecurity defenses through its automated and adaptive rule generation capabilities.

7. Conclusions and Future Work

In this paper, we propose a generative AI-based framework to enhance and secure WAFs against XSS attacks by achieving two main objectives: generating obfuscated XSS payloads using LLMs with curated examples validated against real vulnerable applications, and automating defense mechanisms through machine learning and LLMs to classify bypassing attacks and generate new WAF rules. Our experiments demon- strated the framework's effectiveness generating 220 new XSS payloads by GPT-

4o, 80% of which bypassed state-of-the-art ModSecurity rules. With just 15 new security rules, 83% of these unique attacks were successfully blocked.

7.1. Future Work

Our framework's results open the door to several promising directions for future research and development. Some limitations, such as recall and dataset imbalance, can be mitigated by manually refining the prompt and iteratively testing it. However, as future work, the plan is to leverage agentic AI to automate this process. By implementing a multi-agent system, each agent will handle a specific task generating, testing, or refining prompts and payloads based on incorrect outputs. These agents will communicate with each other to enhance overall performance [17]. Furthermore, ethical AI safeguards, including controlled dataset access and responsible disclosure policies, will ensure that generated attack payloads are used exclusively for defensive cybersecurity applications. Another avenue is training and improving machine learning models for anomaly detection by leveraging multi-agent to preprocess training data and design neural network models [18].

References

1. OWASP Foundation. Cross-Site Scripting (XSS). https://owasp.org/www-community/attacks/xss/. Accessed: 2025-01-07.
2. Applebaum, S.; Gaber, T.; Ahmed, A. Signature-based and machine-learning-based web application firewalls: a short survey. *Procedia Computer Science* **2021**, *189*, 359–367.
3. Mallick, M.A.I.; Nath, R. Navigating the Cyber security Landscape: A Comprehensive Review of Cyber-Attacks, Emerging Trends, and Recent Developments. *World Scientific News* **2024**, *190*, 1–69.
4. Abshari, D.; Fu, C.; Sridhar, M. LLM-assisted Physical Invariant Extraction for Cyber-Physical Systems Anomaly Detection. *arXiv preprint arXiv:2411.10918* **2024**.
5. Zibaeirad, A.; Koleini, F.; Bi, S.; Hou, T.; Wang, T. A comprehensive survey on the security of smart grid: Challenges, mitigations, and future research opportunities. *arXiv preprint arXiv:2407.07966* **2024**.
6. Abshari, D.; Sridhar, M. A Survey of Anomaly Detection in Cyber-Physical Systems. *arXiv preprint arXiv:2502.13256* **2025**.
7. Babaey, V.; Ravindran, A. GenSQLi: A Generative Artificial Intelligence Framework for Automatically Securing Web Application Firewalls Against Structured Query Language Injection Attacks. *Future Internet* **2025**, *17*, 8.
8. Foundation, O. OWASP ModSecurity Core Rule Set Project, 2024. Accessed: 2024-12-03.
9. Wu, C.; Chen, J.; Zhu, S.; Feng, W.; He, K.; Du, R.; Xiang, Y. Wafbooster: Automatic boosting of waf security against mutated malicious payloads. *IEEE Transactions on Dependable and Secure Computing* **2024**.
10. Yao, Y.; He, J.; Li, T.; Wang, Y.; Lan, X.; Li, Y. An Automatic XSS Attack Vector Generation Method Based on the Improved Dueling DDQN Algorithm. *IEEE Transactions on Dependable and Secure Computing* **2023**.
11. Garn, B.; Lang, D.S.; Leithner, M.; Kuhn, D.R.; Kacker, R.; Simos, D.E. Combinatorially xssing web application firewalls. 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2021, pp. 85–94.
12. Alaoui, R.L.; others. Generative Adversarial Network-based Approach for Automated Generation of Adversarial Attacks Against a Deep-Learning based XSS Attack Detection Model. *International Journal of Advanced Computer Science and Applications* **2023**, *14*.
13. Khan, S. LL-XSS: End-to-End Generative Model-based XSS Payload Creation. 2024 21st Learning and Technology Conference (L&T). IEEE, 2024, pp. 121–126.
14. BruteLogic. XSS Gym - p04. Available online: https://brutelogic.com.br/gym.php?p04=red (accessed on 2 December 2024), 2024.
15. Bafna, P.; Pramod, D.; Vaidya, A. Document clustering: TF-IDF approach. 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT). IEEE, 2016, pp. 61–66.
16. Schubert, E.; Sander, J.; Ester, M.; Kriegel, H.P.; Xu, X. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)* **2017**, *42*, 1–21.

17. Singh, A.; Ehtesham, A.; Kumar, S.; Khoei, T.T. Enhancing ai systems with agentic workflows patterns in large language model. 2024 IEEE World AI IoT Congress (AIIoT). IEEE, 2024, pp. 527–532.
18. Et-Tolba, M.; Hanin, C.; Belmekki, A. DL-Based XSS Attack Detection Approach Using LSTM Neural Network with Word Embeddings. 2024 11th International Conference on Wireless Networks and Mobile Communications (WINCOM). IEEE, 2024, pp. 1–6.