The ISC Int'l Journal of Information Security
ISeCure

October 2025, Volume 17, Number 3 (pp. 1–10)
http://www.isecure-journal.org

Presented at the ISCISC'2025 in TEHRAN, Iran.

Genetic Algorithms in Action: Adversarial Attacks on Machine Learning-Based XSS Detection Systems **

Mohammadreza Safi [1] and Mohammad Ali Hadavi [1,*]
[1] *Faculty of Electrical and Computer Engineering, Malek Ashtar University of Technology, Iran*

ARTICLE INFO.

Keywords:
IMachine learning, Adversarial attack, Genetic Algorithms, XSS detection, Cross-Site Scripting

Type:

doi:

ABSTRACT

Cross-Site Scripting (XSS) remains a critical web application vulnerability, consistently ranking among the OWASP Top 10 security risks. Although machine learning and deep learning techniques have improved XSS detection, these models are susceptible to adversarial attacks — carefully crafted inputs designed to evade detection. This paper proposes a novel adversarial attack framework that leverages a Genetic Algorithm to generate adversarial XSS payloads targeting machine learning-based detection systems automatically. Our framework is designed to achieve high transferability, enabling adversarial samples to bypass a wide range of detection models, even those with different architectures. By employing genetic operators such as selection, crossover, and mutation, the framework systematically optimizes payloads to maximize their ability to evade detection while preserving syntactic validity. Experimental results demonstrate that the generated adversarial samples consistently evade multiple state-of-the-art detection models, revealing significant vulnerabilities in current XSS defences. This work underscores the urgent need for more robust machine learning-based security solutions and provides a foundation for developing improved defences against adaptive adversarial threats.

1 Introduction

Cross-Site Scripting (XSS) has consistently been recognised as one of the most significant security vulnerabilities in web applications, prominently featured in the OWASP Top 10 list for decades. XSS attacks exploit inadequate input validation mechanisms, enabling attackers to inject malicious scripts into trusted websites. These scripts execute in the browsers of unsuspecting users, leading to severe consequences such as session hijacking, data theft, and unauthorised actions [1]. Persistent prevalence of XSS underscores the urgent need for robust detection strategies. In response, machine learning (ML) and deep learning (DL) techniques have emerged as promising tools for automated XSS detection. These methods leverage large-scale datasets but are not immune to sophisticated threats. Research has shown that adversarial examples— malicious inputs subtly modified to evade detection—can significantly degrade model accuracy, raising concerns about the re-

* Corresponding author.
**The ISCISC'2025 program committee effort is highly acknowledged for reviewing this paper.
Email addresses: mosafi@mut.ac.ir, hadavi@mut.ac.ir
ISSN: 2008-2045

liability of current defence mechanisms [2]. However, despite their performance advantages, ML-based detectors are not a replacement for validation and output encoding mechanisms.

This paper introduces an innovative adversarial attack framework leveraging Genetic Algorithms (GAs) to generate evasion-capable XSS payloads against ML-based detection systems systematically. Our approach formulates adversarial example generation as an optimization problem, where GAs evolve syntactically valid payloads through iterative selection, crossover, and mutation operations.

A key contribution of this work lies in the high transferability of the generated adversarial samples—a feature that has been less explored in prior research on XSS evasion attacks. Unlike many existing approaches that target specific models or require white-box access, our method produces adversarial inputs that can bypass diverse ML architectures—including CNNs, LSTMs, SVMs, and k-NN classifiers—without prior knowledge of the target system. This characteristic exposes critical vulnerabilities across various detection paradigms and demonstrates the adaptability of evolution-based attack strategies in real-world scenarios. By emphasising transferability in black-box scenarios, this study fills a gap in the current literature and provides valuable insights for developing more resilient ML-based XSS detection systems. The experimental results confirm the effectiveness of our framework in evading multiple state-of-the-art models, reinforcing the necessity of proactive adversarial training and robust model hardening to counter evolving threats.

The main contributions of our work are as follows:

- We propose an adversarial attack framework based on Genetic Algorithm (GA) that automatically generates syntactically valid XSS payloads capable of evading ML-based detection systems. Our framework leverages evolutionary computation — through selection, crossover, and mutation operations — to iteratively refine malicious inputs and maximise evasion success while maintaining functional validity.
- Our approach is fully model-agnostic, successfully bypassing a wide range of ML/DL architectures — including CNNs, LSTMs, RNNs, SVMs, k-NN classifiers, and Naïve Bayes — without requiring prior knowledge of the target model's structure or parameters. This work demonstrates strong transferability and cross-model generalizability, highlighting the real-world applicability of the proposed method under adaptive black-box attack scenarios.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 and Section 4 present the proposed method and experimental results, respectively. Section 5 concludes the paper.

2 Related Work

In this section, we review recent research on black-box adversarial attacks against XSS detection systems.

Wang *et al.* [3] proposed a fuzzy-based approach to achieve a "white-box attack" to improve the confidence level of adversarial samples effectively. Additionally, this article presents a reinforcement learning-based adversarial attack model designed to generate adversarial attack samples for various XSS attack detection models using multiple strategies. Experimental results show that the proposed attack model generates adversarial samples against different XSS attack detection models with an evasion rate of over 85%

Fang *et al.* [4] presented a reinforcement learning-based method to optimising the XSS detection model to defend against adversarial attacks. Initially, adversarial samples from the detection model are extracted using an adversarial attack model based on reinforcement learning. Secondly, the detection and adversarial model are trained alternately. After each round, the newly generated adversarial samples are labeled as malicious and used for retraining the detection model. Experimental results show that the proposed model can successfully extract adversarial samples that evade both black-box and white box detection. Experimental results demonstrate that alternating the training of the detection model enhances its robustness against adversarial samples generated during the process.

In [5], the authors present a novel approach that integrates ML and DL frameworks to detect and defend against XSS attacks effectively. The method uses four datasets with both safe and unsafe content to adaptively identify URL-based attacks, while enhancing prediction accuracy and training speed through URL encoding and dictionary-based mapping to remove unsafe JScript/JavaScript keywords.

Recent studies have demonstrated that genetic algorithms are effective in generating XSS attack payloads and optimising feature selection for machine learning–based detection models. For instance, the GAXSS method utilises genetic algorithm–based fuzzing framework to automatically generate high-quality attack vectors, achieving high precision and accuracy rates in vulnerability detection [6]. Similarly, the GeneMiner employs incremental genetic algorithm to enhance classification accuracy in identifying malicious payloads, showcasing a significant

improvement over conventional detection techniques [7].

A key limitation observed in previous studies is the lack of cross-model transferability in the generated adversarial samples. Most existing works focus on crafting payloads that are optimised for a specific detection model and show reduced effectiveness when applied to different architectures. In our work, we address this limitation by designing a framework that systematically evaluates and enhances the transferability of adversarial XSS payloads across multiple ML/DL models.

3 PROPOSED METHOD

In this section, we present a novel adversarial attack framework designed for generating adversarial samples tailored for ML-based XSS detection systems. Our approach iteratively evolves malicious payloads to simultaneously ensure syntactic correctness and functional executability, while maximising evasion effectiveness across a diverse set of ML-based detection systems.

3.1 Solution Overview

We use GAs to generate syntactically valid XSS payloads that evade ML-based detection systems. Unlike traditional fuzzing or rule-based obfuscation techniques, our method systematically evolves malicious inputs through an iterative optimisation process that balances evasion success with payload execution functionality.

Figure 1 presents the general architecture of the proposed adversarial attack framework based on GAs. Our framework integrates several key components to systematically evolve malicious payloads while preserving their exploitability.

- Payload Structure: Each XSS payload is composed of HTML tags, attributes, event handlers, and executable JavaScript code.
- Fitness Evaluation: Payloads are evaluated based on their ability to evade detection, structural complexity, and syntactic validity.
- Evolution Process: The framework iteratively refines candidate payloads through mutation and crossover operations.
- Detection Interface: Generated payloads are evaluated against ML-based XSS detectors in a black-box setting, with feedback used to guide subsequent evolutionary iterations.

The use of GAs is particularly well-suited for this task due to several advantages:

- Black-box compatibility: No internal knowledge of the target model is required; only input-output feedback is used.
- Exploration of large solution space: GAs efficiently explore complex combinations of HTML tags, event handlers, and JavaScript constructs.
- Preservation of functional behaviour: Through structured representation and semantic-aware mutation, evolved payloads retain their malicious intent.
- Adaptability: The algorithm dynamically adjusts mutation rates and obfuscation patterns based on fitness feedback, enabling robust evasion across diverse models.

Our approach systematically addresses the challenge of generating effective adversarial XSS samples under black-box constraints. Detailed implementation of each component— payload structure, mutation strategies, and fitness function—is described in the subsequent subsections.

3.2 XSS Escape Structure

The structure of common XSS attack vectors can be effectively analyzed and utilized in the context of genetic algorithms for vulnerability detection.

As shown in Figure 2, this approach involves understanding the components that make up these attack vectors, which typically consist of:

Tags: HTML elements such as `<script>`, `<img>`, and `<body>`, which can be manipulated to execute malicious scripts.

Attribute Expressions: Attributes within HTML tags that can trigger JavaScript execution, such as `src` and `href`.

Event Expressions: JavaScript events that can execute code when triggered by certain actions, such as `onload` and `onerror`.

Executable Code Content: The JavaScript code intended for execution, which may include payloads specifically crafted to exploit vulnerabilities.

Closed Tags: Properly formatted HTML tags that encapsulate the above elements, ensuring that the browser interprets them correctly.

This division into components facilitates the implementation of genetic operators within the framework. In this context, chromosomes are represented as combinations of these elements, allowing generation and optimisation of attack vectors through evolutionary processes.

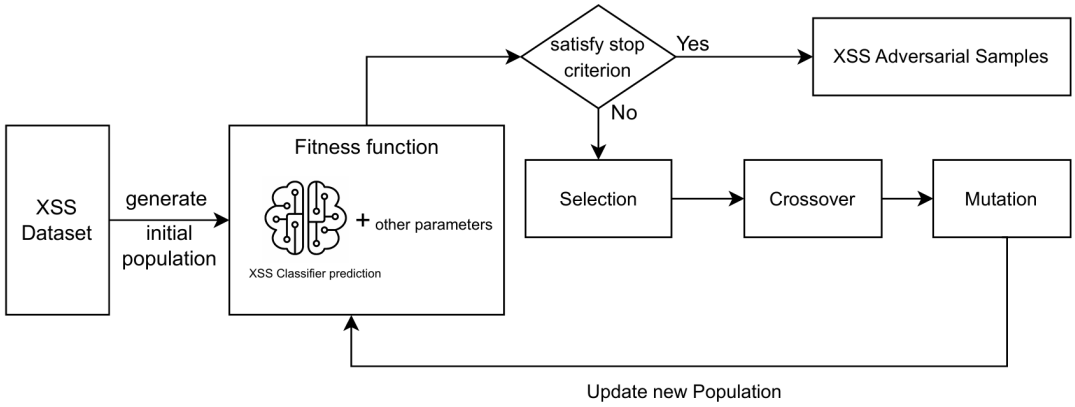

Figure 1. The adversarial attack framework

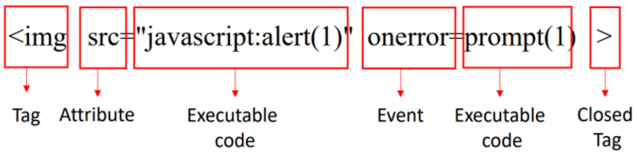

Figure 2. XSS Payload Structure

3.3 Attack Framework

The Initial population is randomly sampled from the dataset. For each individual, a fitness value is computed to evaluate how well it addresses the target problem. Fitness is primarily determined by feedback from XSS detection models, while the remaining parameters have only a marginal impact on the fitness score. After computing fitness values, individuals are ranked, and the top-ranked samples are selected to form the next generation. This selection process favours higher-fitness individuals, increasing the likelihood that their genetic material will be passed on to subsequent generations.

Crossover and mutation are key genetic operators used to generate new populations and improve solution quality. These operators increase the likelihood of producing high-quality adversarial payloads, and the evolutionary process continues until termination conditions are met, such as reaching the maximum number of generations or achieving convergence.

The pseudo-code of our proposed method is given in Algorithm 1. As described in this pseudo-code, our attack framework includes the typical operators in GAs: initialisation (line 1), fitness evaluation (line 5), selection (line 11), crossover (line 14), update population (line 16) and mutation (line 18).

Initialization. The initial population plays a critical role in the evolutionary process, influencing both convergence behaviour and the quality of final adversarial samples. A well-chosen initial population ensures diversity, which is essential for exploring the complex space of HTML/JavaScript-based XSS attacks.

In our framework, the initial population is constructed by randomly sampling syntactically valid XSS payloads from a curated dataset of real-world attack vectors. These samples are drawn from known XSS injection points and include a variety of event handlers, attributes, and obfuscation techniques to promote diversity and exploitability. This random initialisation strategy helps prevent early convergence to suboptimal solutions and allows the algorithm to discover novel evasion patterns in subsequent generations.

Fitness Evaluation. The fitness score of a payload can be evaluated based on the following factors:

(1) The major part of the fitness function is based on the estimation and feedback of the detection model for XSS. In detection model, if the estimation is zero, the payload is classified as benign (non-malicious). Function $F_1(p)$, defined in the following relation, represents this part of our fitness function.

$$F_1(p) = (1 - XSSClassifier.predict(p)) \quad (1)$$

(2) The increased use of labels, events, and attributes indicates a more complex payload, which raises the likelihood of misclassification by the detection model. Accordingly, number of events and tags in each payload is calculated and weighted by a coefficient in the fitness function. Let n be the total number of possible events and m the total number of possible tags in a payload p. Then, $E(p)$ and $T(p)$ represent the total numbers of events and tags used in p, respectively, where $Event_i$ and Tag_j are binary indicators that return 1 if the corresponding event or tag exists in p, and 0 otherwise.

Algorithm 1 Proposed Genetic Algorithm

Require: mutation-rate mr, crossover-rate cr, generation G, xssDataset D, population-size N, matingPool-rate mpr, xssClassifier cls
Ensure: Adversarial example list adv_list

```
pop ← init_population(D, N)
g ← 0
while g < G and AverageFitness(pop) < 1 do
    for each x in pop do
        fitness_calc(x, cls)
        if x.fitness ≥ 1 then
            adv_list.append(x)
        end if
    end for
    sort_by_fitness(pop)
    mating_pool ← selection(pop, mpr, N)
    for i = 0 → (N × mpr) do
        p1, p2 ← random_parent(mating_pool)
        children ← crossover(p1, p2, cr)
    end for
    pop ← update_population(pop, children)
    for each x in pop do
        mutation(x, mr)
    end for
    g ← g + 1
end while
return adv_list
```

$$E(p) = \sum_{i=1}^{n} Event_i \tag{2}$$

$$T(p) = \sum_{j=1}^{m} Tag_j \tag{3}$$

Using the definitions of $E(p)$ and $T(p)$, the formula for this part of the fitness function, $F_2(p)$, is expressed as below:

$$F_2(p) = (E(p) + T(p)) \times 0.01 \tag{4}$$

(3) The payload's structure is critical for exploiting XSS vulnerabilities and ensuring the success of the adversarial sample. To ensure the proper exploration of the solution space and the generation of new solutions using genetic operators, it is crucial to validate the syntactical correctness of payloads. In our work, the initial validation focuses on the correct placement of attributes and event handlers within HTML elements. This is essential for crafting syntactically valid adversarial samples. In fact, Attributes and Events must be included within the opening tag of an HTML element. The syntax penalty F_3 is intentionally designed as a control mechanism to immediately penalize syntactically invalid payloads without affecting the primary optimisation driven by F_1. The correctness of syntactical rules is evaluated using $F_3(p)$ defined as below:

$$F_3(p) = \begin{cases} -0.5, & \text{if syntactical rules are violated} \\ 0, & \text{if syntactical rules are followed} \end{cases}$$

Combining the effects of the above factors, the overall fitness function $F(p)$ for a payload p is defined as:

$$F(p) = F_1(p) + F_2(p) + F_3(p) \tag{5}$$

Selection. The selection method described here is a hybrid approach used in GAs to balance the preservation of high-quality solutions with the introduction of genetic diversity. We select 80% of the new generation samples from the most fit individuals in the current population, ensuring that the best-performing solutions are carried forward to the next generation. This approach accelerates convergence toward optimal solutions by prioritizing traits that improve performance.

The remaining 20% of the new generation samples are randomly selected from the rest of the population, which includes less fit individuals. Random selection introduces genetic diversity into the population, preventing premature convergence and allowing for the exploration of new and potentially beneficial traits that may not be present in the fittest individuals. By maintaining a diverse gene pool, this method enhances the algorithm's ability to explore the solution space more thoroughly, increasing the chances of discovering innovative solutions.

Crossover. This mechanism is crucial for exploring the solution space and generating new solutions that may exhibit improved characteristics. In our problem, the crossover rate and the length of the smallest payload in the population are first determined. The crossover rate is multiplied by the length of the smaller payload to calculate an integer value, which determines the number of genes to be exchanged between the parents. Based on this value, a random selection of genes is swapped between the two parents, resulting in two offspring. Each offspring inherits a portion of the genetic information from both parents, ensuring diversity and promoting the exploration of new solutions within the population.

Mutation. In our work, the mutation rate plays a critical role in maintaining genetic diversity and enabling exploration of the solution space. A dynamic mutation rate that adjusts according to the population's average fitness score can enhance the algorithm's ability to escape local optima and improve overall solution quality. The algorithm starts with an initial mutation rate, typically set to a small per-

centage. The average fitness score of the population is calculated at the end of each generation. After evaluating the fitness of all individuals in the population, the average fitness score is computed. This score serves as a benchmark for assessing the effectiveness of the current population.

The current average fitness score is compared with that of the previous generation to assess whether the population is improving or stagnating. A decrease in the average fitness score indicates that the population may be converging prematurely or failing to explore new solutions effectively. In such cases, the mutation rate is incrementally increased to introduce greater diversity, promote broader exploration, and help the algorithm escape local optima. This adaptive strategy enables gradual adjustments while preserving fine-grained control over the evolutionary process. The mutation strategies employed in our framework are primarily derived from OWASP XSS attack techniques [8]:

- HTML allows the use of custom tags (also known as "non-standard" or "unknown" tags) which can execute certain specific events within them. for example:
  `<custom-tag onclick=alert(1)></custom-tag>`
- Adding tags inside other tags or outside of them increases the complexity of the payload. The increased complexity of the payloads decreases the probability of their identification as malicious.
- By adding or replacing attributes when filtering items, the probability of identification by the classifier can be reduced.
- Random characters are added to the payload to make it less recognizable to simple pattern-matching filters.
- Use a slash between the tag and its attribute without a space.
- When specific events are detected or filtered, one approach is to replace them with other events, such as substituting `onclick` events with `onload` events.
- Adding single-line comments using double slashes.

By using a dynamic mutation rate that adapts to average fitness changes, our algorithm balances exploration and exploitation, avoids premature convergence, and achieves better overall performance.

Update Population. The new generation is evaluated by fitness, and the top individuals from both old and new populations are selected to form the updated population, ensuring only the best candidates proceed to the next iteration.

Termination condition. The evolutionary process terminates when either:

(1) Limitations on the number of generations, with a maximum of 20 generations considered. It is important to note that this also depends on the size of the initial population and the offspring population in each generation. Variations in these parameters result in proportional adjustments to the generation limit. Here, the initial population is 128 individuals.
(2) The average fitness value is equal to or greater than 1. In this situation, all payloads in the population have reached the maximum fitness value. The average fitness is computed as follows:

$$\text{AverageFitness} = \frac{1}{N}\sum_{i=1}^{N}\text{Fitness}(i) \tag{6}$$

Where:

- AverageFitness is the average fitness value of the population.
- N is the total number of individuals in the population.
- Fitness(i) is the fitness score of the i-th individual in the population.

In the next section, we delineate the step-by-step implementation of our algorithm.

4 Experiments

Dataset. In our study, we utilised a comprehensive dataset obtained from several sources, including GitHub repositories and Kaggle datasets [9, 10]. Our dataset contains approximately 32,000 malicious and benign JavaScript payloads.

Experimental Setup. Table 1 displays the settings of the simulation and evaluation environment.

Table 1. Experimental Environment and Configuration

Parameter	Value
Environment	Google Colab
Processor	Intel Xeon CPU
RAM	12GB
Python version	3.8
API	Keras (based on TensorFlow)
Datasets	GitHub [9], Kaggle [10]

Classification Models. Table 2 and Table 3 summarise the architectures and hyperparameters of our models, covering both deep learning approaches (CharCNN, RNN, LSTM) and traditional classifiers (NB, SVM, KNN). To preprocess the input, we employed character-level tokenisation; each character

in a payload was mapped to a unique integer index derived from the dataset vocabulary. The resulting sequences were padded or truncated to a fixed maximum length of 512, ensuring uniform input dimensions across all models. This design not only provides comparability between model families but also captures subtle, fine-grained character patterns that are particularly relevant for detecting adversarial XSS payloads.

Classifier results. Table 4 presents the performance metrics for various classification algorithms, including RNN, LSTM, CNN, KNN, Naïve Bayes, and SVM. Notably, all evaluation metrics exceed 97%, indicating that each algorithm achieves high classification accuracy. These performance metrics are critical for assessing how well each algorithm withstands adversarial Perturbations designed to deceive the model.

Adversarial Attack. Hyperparameters are key factors in adversarial attack frameworks, directly affecting the efficiency of generating adversarial examples. In this study, the genetic algorithm is configured with the parameters listed in Table 5 for up to 20 generations, balancing exploration and exploitation to ensure effective convergence.

Figure 3 shows the average fitness values across generations, highlighting how the GA-based attack steadily enhances payload quality. Individuals with higher fitness, representing better evasion, are preferentially reproduced, leading to consistent improvement and demonstrating the algorithm's ability to adapt and bypass detection mechanisms.

Across all evaluated models, the average fitness value converges after approximately 20 generations, indicating solution stability and suggesting near-optimal performance. This convergence pattern demonstrates that further iterations would yield minimal improvements, empirically validating our termination condition's effectiveness in balancing solution quality with computational efficiency.

Table 6 provides a comprehensive summary of the final values of the fitness function achieved across generations for different classifiers. All models achieved a best fitness score greater than 1, indicating that at least one payload successfully evaded each detection model. These results demonstrate the effectiveness of our framework in generating adversarial XSS samples capable of bypassing various ML-based detectors.

Figure 4 illustrates the results of the proposed adversarial attack framework over a series of 20 generations, showcasing the cumulative number of unique payloads extracted for each classification model tested. Each bar represents a different ML model employed in XSS detection, highlighting the varying effectiveness of the adversarial attack across these models. The results indicate that the k-NN model yielded the highest number of extracted payloads, with a total of 934, suggesting that this model may be particularly vulnerable to adversarial strategies employed. In contrast, for the LSTM model 349 payloads were generated.

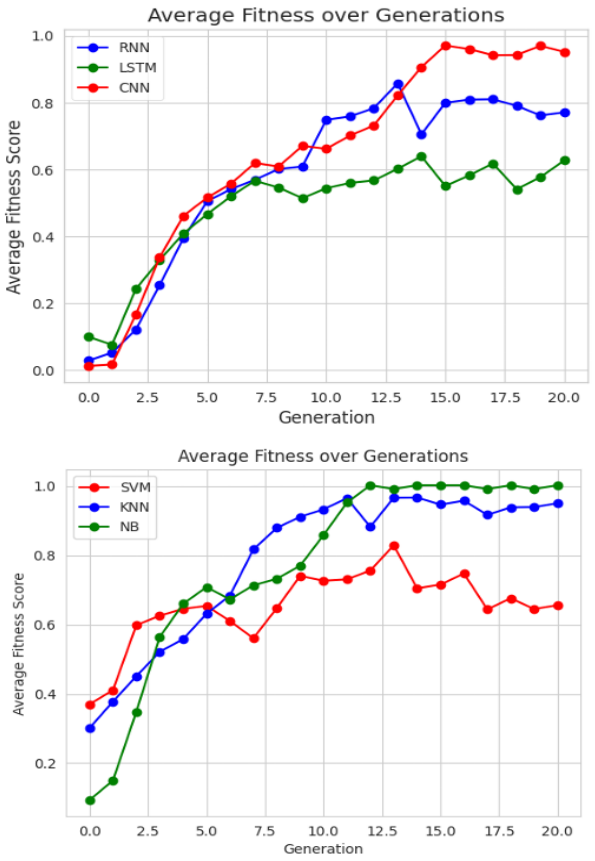

Figure 3. The average value of the fitness function per generation in adversarial attack on each classifier model

Evasion Rate. The evasion rate is a metric used to evaluate the effectiveness of an attack. It measures the proportion of adversarial samples that successfully bypass a detection system. The formula for calculating the evasion rate can be expressed as follows:

$$\text{Evasion rate} = \frac{\text{Number of Successful Evasions}}{\text{Total Number of Attacks}}$$

where:

- NumberofSuccessfulEvasions is the number of adversarial samples that evade the detection system.
- TotalNumberofAttacks is the total number of adversarial samples generated and tested against the detection system.

Transferability Evaluation. Transferability is a critical property of adversarial examples, particularly in black-box attack scenarios. It refers to the

Table 2. Architectural configuration of CharCNN, RNN, and LSTM models

Layers	CNN	RNN	LSTM
Embedding	input_dim=73, output_dim=80, input_length=512, trainable=False	input_dim=73, output_dim=128, input_length=512	input_dim=73, output_dim=128, input_length=512
Core Layers	Conv1D: filters=32, kernel_size=[4,8], activation='ReLU' MaxPooling1D: pool_size=[12,11]	SimpleRNN: units=64	LSTM: units=128, return_sequences=True
Dense Layers	Dense: 256 (ReLU) Dense: 128 (ReLU) Dense: 1 (Sigmoid)	Dense: 1 (Sigmoid)	Dense: 1 (Sigmoid)
Dropout Rate	0.3	0.5	0.5

Note: All models use binary cross-entropy loss and Adam optimizer.

Table 3. Main hyperparameters of ML-Base classifiers

Model	Hyperparameters
NB	`alpha=1.0`
SVM	`kernel=linear, C=1.0`
KNN	`n_neighbors=5, metric=euclidean`

Table 4. Accuracy, precision, recall, F1-Score values for the classification algorithms

Model	Accuracy	Precision	F1-Score	Recall
CNN	0.999	0.999	0.999	0.999
LSTM	0.997	0.999	0.996	0.994
RNN	0.997	0.999	0.996	0.994
SVM	0.994	0.996	0.994	0.993
KNN	0.968	0.987	0.969	0.952
NB	0.968	0.949	0.970	0.993

Table 5. Hyperparameters of genetic algorithm

Mutation rate	Crossover rate	Mating-pool rate	Population size
0.1	0.3	0.5	128

Table 6. Final fitness values over 20 generations

Model	Best	Average	Lowest
CNN	1.04	0.94	0.02
LSTM	1.03	0.64	-0.29
RNN	1.08	0.78	0.48
SVM	1.01	0.66	0.02
KNN	1.01	0.95	0.05
NB	1.06	0.99	0.02

ability of an adversarial sample crafted to evade one ML model to also successfully bypass other models — even those with different architectures or training data. In this study, we evaluate the transferability of the adversarial XSS payloads generated by our Framework across multiple ML/DL models. Specifically, we

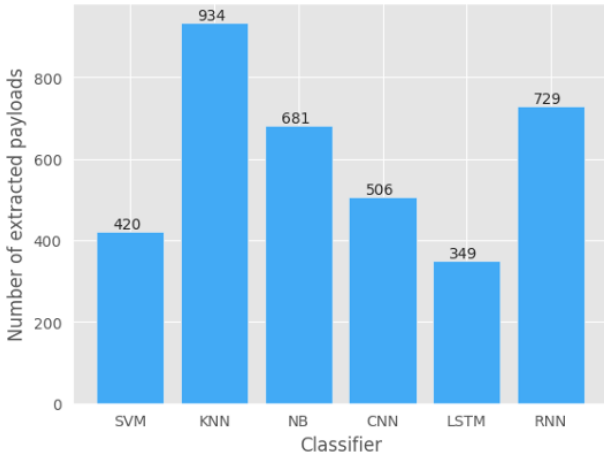

Figure 4. Cumulative number of unique extracted payloads from the proposed attack framework (in the 20th generation) for each classification model

selected 300 adversarial samples (50 from each source model) and tested them against six target classifiers: CNN, LSTM, RNN, SVM, k-NN, and Naïve Bayes.

Table 7 presents the evasion results for each target model, where: NoX denote number of adversarial payloads not detected (i.e., successfully evaded). The results demonstrate that the adversarial payloads generated by our method exhibit strong cross-model evasion capabilities. Notably, the highest evasion rate of 75% was achieved against the k-NN classifier, while Naïve Bayes and CNN followed with 48% and 43%, respectively.

These findings highlight the generalization power of our approach, indicating that the evolved payloads are not optimized solely for a specific model but instead exploit broader patterns common across different detection systems. This is largely attributed to our domain-aware mutation strategies and multi-objective fitness function, which encourage diversity and syntactic validity in payload generation. This experiment underscores the vulnerability of current ML-

Table 7. Evasion rate

Target Model	NoX	ER
CNN	129	43%
LSTM	83	27%
RNN	94	31%
SVM	73	24%
KNN	226	75%
NB	145	48%

based XSS detectors to adversarial attacks and emphasises the need for more robust defence mechanisms capable of handling adaptive evasion techniques.

5 Conclusion

In this work, we presented a novel adversarial attack framework based on a GA to bypass ML-based XSS detection systems. In this research, we aimed to use GA to transform the problem of finding adversarial examples for Script injection into an optimisation problem. This problem focuses on discovering malicious payloads that can successfully bypass or deceive black-box machine learning models, even in the absence of information about the models' learning type and internal structure. Through extensive experiments, we demonstrate that our approach effectively crafts adversarial payloads capable of evading ML-based XSS detection models with high success rates. These results highlight the vulnerability of current ML-based XSS detection systems to carefully designed adversarial attacks and the need for more robust defence mechanisms. We also demonstrated that the proposed adversarial attack framework exhibits high transferability, effectively bypassing various models even when they differ in architecture.

Moving forward, this research opens several important directions for future work, including improving the generalization of our adversarial payloads and developing more effective defence strategies (e.g., adversarial training and detection model hardening), and exploring the transferability of our approach to other security domains beyond XSS detection. Overall, this work contributes to the growing field of adversarial ML, emphasising the importance of considering adversarial threats when designing and deploying ML-based security systems. Continued research in this area is crucial to developing more robust and secure solutions.

References

[1] OWASP Foundation. Cross Site Scripting (XSS), 2025. URL https://owasp.org/www-community/attacks/xss/. Accessed January 2025.

[2] OWASP Foundation. The OWASP Top Ten 2025, 2025. URL https://www.owasptopten.org/. Accessed January 2025.

[3] Qiuhua Wang, Hui Yang, Guohua Wu, Kim-Kwang Raymond Choo, Zheng Zhang, Gongxun Miao, and Yizhi Ren. Black-box adversarial attacks on xss attack detection model. *Computers & Security*, 113:102554, 2022.

[4] Yong Fang, Cheng Huang, Yijia Xu, and Yang Li. Rlxss: Optimizing xss detection model to defend against adversarial attacks based on reinforcement learning. *Future Internet*, 11(8):177, 2019.

[5] Muralitharan Krishnan, Yongdo Lim, Seethalakshmi Perumal, and Gayathri Palanisamy. Detection and defending the xss attack using novel hybrid stacking ensemble learning-based dnn approach. *Digital Communications and Networks*, 10(3):716–727, 2024.

[6] Zhonglin Liu, Yong Fang, Cheng Huang, and Yijia Xu. Gaxss: effective payload generation method to detect xss vulnerabilities based on genetic algorithm. *Security and Communication Networks*, 2022(1):2031924, 2022.

[7] Charu Gupta, Rakesh Kumar Singh, and Amar Kumar Mohapatra. Geneminer: a classification approach for detection of xss attacks on web services. *Computational Intelligence and Neuroscience*, 2022(1):3675821, 2022.

[8] OWASP Foundation. XSS Filter Evasion Cheat Sheet, 2024. URL https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html. Accessed May 2024.

[9] GitHub. XSS Payload, Source URL, 2024. URL https://github.com/payloadbox/xss-payload-list/blob/master/Intruder/xss-payload-list.txt. Accessed May 2024.

[10] Kaggle. XSS Data, Source Classification, Online URL, 2024. URL https://www.kaggle.com/code/hafizfarhad/cross-site-scripting-xss-prediction/input?select=XSS_dataset.csv. Accessed October 2024.

Mohamadreza Safi received his B.Sc degree in computer Engineering from iran University of science and technology, Tehran, Iran in 2021. He then received his M.Sc degree in Secure Computing from Malek Ashtar University of Technology, Tehran, Iran, in 2024. His research interests include adversarial machine learning and Cloud Computing security.

Mohammad-Ali Hadavi received his PhD in Computer Engineering from Sharif University of Technology, Tehran, Iran, in 2015. He received his M.Sc and B.Sc in Software Engineering from Amirkabir University of Technology, Tehran, Iran, in 2004, and from Ferdowsi University of Mashhad, Iran, in 2002, respectively. Now, he is an assistant professor at the Malek Ashtar University of Technology. Focusing on information security, he has published more than 50 papers in national and international journals and conference proceedings. His research interests include software security, machine learning for cyber-security, security measurement, database security, and cloud security.