DeepSQLi-Defense: Integrated Offensive-Defensive AI System for SQL Injection with Adaptive Firewall Learning

Radu-Ionuț Bălăiță, Stefana-Beatrice Gherghel, Ionel-Cătălin Butacu
Faculty of Automatic Control and Computer Engineering
Technical University Gheorghe Asachi Iași
Iași, Romania
{radu-ionut.balaita, stefana-beatrice.gherghel, ionel-catalin.butacu}@student.tuiasi.ro

***Abstract*—SQL Injection (SQLi) remains a critical vulnerability in web applications, often bypassing static firewalls through syntactic obfuscation. This paper presents an integrated offensive-defensive AI system combining three components: (1) DeepSQLi-RL, a hybrid Reinforcement Learning BiLSTM agent for automated penetration testing achieving 99.5% WAF bypass rate, (2) a dual-model ML detector using CNN and RandomForest classifiers for real-time attack identification, and (3) an adaptive firewall that learns from detected attacks using DBSCAN clustering to automatically generate Snort IDS rules. We evaluate the complete pipeline with 150 benchmark payloads, demonstrating 92% attack detection rate with 94.8% F1 score. The adaptive firewall reduces response time by 44% after pattern learning, with pattern-matched blocks averaging 15ms compared to 89ms for ML-based detection. Our results validate the efficacy of combining offensive AI testing with defensive ML detection and unsupervised pattern learning for comprehensive SQLi protection.**

***Index Terms*—SQL injection, reinforcement learning, BiLSTM, adaptive firewall, DBSCAN clustering, machine learning, intrusion detection**

I. Introduction

Web applications are ubiquitous in modern infrastructure, making them prime targets for cyberattacks. Among these, SQL Injection (SQLi) persists as a top threat according to OWASP Top 10 [1], allowing attackers to manipulate backend databases. Defensive mechanisms like Web Application Firewalls (WAFs) typically rely on signature matching (e.g., blocking "UNION SELECT") to prevent attacks. However, attackers constantly evolve methods to evade these rules through obfuscation techniques.

Traditional automated scanners like SQLMap rely on predefined heuristic engines [4]. While effective against basic targets, they often fail against custom WAF rules. Liu et al. [3] introduced DeepSQLi, treating SQLi generation as a Natural Language Processing (NLP) translation task using BiLSTM. Building upon this foundation, we present an integrated system that addresses both the offensive and defensive aspects of SQLi security.

Our contribution is threefold:

1) **DeepSQLi-RL** (Radu): A hybrid extension combining Seq2Seq BiLSTM generation with Q-Learning reinforcement learning for automated penetration testing, achieving 99.5% WAF bypass rate.
2) **ML Detector** (Beatrice): A dual-model classifier using CNN and RandomForest with automatic fallback, providing real-time attack detection with 92% accuracy.
3) **Adaptive Firewall** (Cătălin): An intelligent firewall that learns from ML-detected attacks using DBSCAN clustering and TF-IDF vectorization, automatically generating Snort rules for pattern-based blocking.

The key innovation is the feedback loop between components: the offensive agent tests defenses, the ML detector identifies attacks, and the firewall learns patterns for faster future blocking.

II. Offensive Agent: DeepSQLi-RL

The offensive module implements a hybrid AI pipeline combining generative deep learning with adaptive reinforcement learning for automated penetration testing.

A. BiLSTM Generator

We employ a Seq2Seq architecture with Bidirectional Long Short-Term Memory (BiLSTM) units for payload generation. This architecture captures context from both past and future tokens, making it well-suited for SQL syntax understanding.

- **Encoder**: Processes input payload character-by-character, converting it into a context vector (embedding dimension: 32, hidden dimension: 64).
- **Decoder**: Generates multiple output candidates token-by-token, learning to introduce obfuscations while preserving valid SQL syntax.
- **Diversity**: Generates 7 mutation candidates per input, combining BiLSTM variants with traditional obfuscation techniques.

B. Q-Learning Selector

A tabular Q-Learning agent learns to select the most effective mutation candidate based on WAF responses:

- **State Space**: 192 possible states encoding payload features (length, quotes, comments, keywords, URL encoding).

- **Action Space**: Selection of one candidate from 7 generated mutations.
- **Reward**: $R = +10$ for WAF bypass, -1 for detection.
- **Learning**: Q-table updates via Bellman equation with learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.95$.

C. Offensive Results

We evaluated against a simulated WAF-protected target with blacklist rules for common SQLi signatures (Table I).

TABLE I
WAF BYPASS PERFORMANCE COMPARISON

Method	Bypass Rate	Time/1000
Standard Dictionary	58.7%	12.1s
BiLSTM Only	82.3%	13.6s
Hybrid RL-BiLSTM	**99.5%**	31.3s

The hybrid approach achieves near-perfect bypass rates, representing a 40.8% improvement over standard methods and 17.2% over BiLSTM alone.

III. ML DETECTOR: DUAL-MODEL CLASSIFICATION

The defensive ML module provides real-time attack classification with automatic model fallback for deployment flexibility.

A. Primary Model: CNN Classifier

A Convolutional Neural Network processes character-level tokenized inputs:

- **Input**: Sequences of length $L = 100$ characters
- **Embedding**: 128-dimensional character embeddings
- **Convolution**: Multiple filter sizes (2, 3, 4, 5) with 128 filters each
- **Pooling**: Global max pooling to extract salient features
- **Output**: Sigmoid activation with attack weight boosted by 1.5×

The CNN was trained on 50,000+ SQLi samples with data augmentation. During training, the attack class weight was boosted by 1.5× to bias the model toward security-conscious classification.

B. Fallback Model: RandomForest

When CNN loading fails (e.g., missing TensorFlow), the system falls back to RandomForest:

- **Features**: TF-IDF vectorization with character n-grams (2-5)
- **Estimators**: 100 decision trees
- **Max Features**: 500 TF-IDF features

C. CNN vs RandomForest: Behavioral Analysis

Comparative testing revealed significant behavioral differences between models (Table II).

Key Observations:

- **Confidence Calibration**: CNN produces 100% confidence on clear attacks while returning near-zero (10^{-15}) for safe queries. RF shows less calibrated confidence levels.
- **Generalization**: For known attacks, both models achieve 100% recall on training and test sets. However, RF shows "memorization" behavior on novel inputs.
- **Edge Case**: The nonsense word "supercalifragilisticexpialidocious" reveals model behavior—RF classifies with 100% attack confidence (memorization failure), while CNN shows 95% (reasoned uncertainty).
- **Complex Safe Queries**: CNN correctly classifies SQL-like safe queries (e.g., `UPDATE calculator...`), while RF produces false positives.

TABLE II
MODEL COMPARISON ON EDGE CASES

Query	Label	CNN	RF
`' UNION SELECT 1`	Attack	Attack	Attack (0.6)
`' un%69on se%6cect`	Attack	Attack	Attack
`SE%20LECT * FROM use%72s`	Attack	Attack	Attack (0.7)
UPDATE calc SET result=100 WHERE formula='10*10=100'	Safe	Safe	Attack (0.5)
SELECT WHERE desc LIKE '%0x%'	Safe	Safe	N/A
supercalifragilistic...	Safe	Attack (0.95)	Attack (1.0)

The CNN demonstrates genuine learning by reasoning about unknown inputs, whereas RF relies more heavily on pattern memorization. This validates CNN as the primary model with RF as a lightweight fallback.

D. Detection Threshold

Both models use a configurable threshold $\tau = 0.2$ for classification:

$$\text{Class} = \begin{cases} \text{ATTACK} & \text{if } P(\text{attack}|x) > 0.2 \\ \text{SAFE} & \text{otherwise} \end{cases} \quad (1)$$

The low threshold prioritizes security over convenience, preferring false positives over missed attacks in critical infrastructure contexts.

IV. ADAPTIVE FIREWALL: PATTERN LEARNING

The adaptive firewall represents our main architectural contribution, implementing intelligent pattern learning through unsupervised clustering.

A. System Design

The firewall operates in three configurable modes (Table II), enabling incremental deployment and comparative evaluation.

TABLE III
DEFENSE MODE CONFIGURATIONS

Case	Mode	Active Components
0	Passthrough	None (baseline)
1	ML Only	Detector classifies each request
2	Full Pipeline	Pattern + ML + Learning

B. Defense Pipeline (Case 2)

In full pipeline mode, requests flow through multiple security layers:

1) **Pattern Matching**: Incoming payload is normalized and checked against known attack patterns stored in memory
2) **ML Classification**: If no pattern match, payload is forwarded to the detector API
3) **Learning Trigger**: If ML detects attack (confidence $>$ 0.2), payload is blocked and added to collection
4) **Pattern Extraction**: When collection reaches threshold (5+ payloads), DBSCAN clustering extracts common patterns
5) **Rule Generation**: New patterns are converted to Snort IDS rules for permanent blocking

C. Pattern Extraction Algorithm

The `AttackPatternExtractor` class implements a multi-stage extraction pipeline:

1) TF-IDF Vectorization: Payloads are converted to numerical vectors using character n-grams:

$$\text{TF-IDF}(t, d, D) = \text{tf}(t, d) \cdot \log \frac{|D|}{|\{d \in D : t \in d\}|} \quad (2)$$

where t is a character n-gram (2-4 characters), d is a payload, and D is the corpus. This captures SQL syntax patterns while maintaining computational efficiency.

2) DBSCAN Clustering: Similar payloads are grouped using density-based clustering:

- Distance threshold (ϵ): 0.5
- Minimum cluster size: 5 payloads
- Distance metric: Cosine similarity

Unlike k-means, DBSCAN automatically determines cluster count and identifies outliers (novel attacks that don't fit existing patterns).

3) Frequent Substring Mining: Within clusters, common substrings appearing in $>$10% of payloads are extracted. The algorithm prioritizes longer patterns to minimize false positives:

for each length l from 20 down to 3 **do**
  for each payload p in cluster **do**
    Extract all substrings of length l
    Count frequency across cluster
  end for
  Keep substrings with frequency $> 10\%$
end for

D. Payload Normalization

Before pattern matching, payloads undergo normalization to handle common evasion techniques:

$$\text{norm}(p) = \text{collapse\_ws}(\text{rm\_comments}(\text{lower}(p))) \quad (3)$$

This handles:

- Case variation: `UNION` vs `union`
- Comment injection: `UN/**/ION` $\rightarrow$ `UNION`
- Whitespace manipulation: Multiple spaces $\rightarrow$ single space

E. Snort Rule Generation

Extracted patterns are automatically converted to Snort IDS rules:

Listing 1. Auto-Generated Snort Rule

```
alert tcp any any -> any any (
  msg:"AI-Learned: UNION SELECT";
  flow:to_server,established;
  content:"UNION SELECT"; nocase;
  classtype:web-application-attack;
  sid:9021685; rev:1;
  metadata:created 20260115;)
```

During benchmark testing, the firewall generated 33 unique Snort rules from learned attack patterns.

V. Experimental Evaluation

A. Testbed Configuration

The system was deployed in a virtualized environment:

- **Defense VM**: Ubuntu 22.04, hosting Detector (port 5000) and Firewall (port 5001)
- **Target VM**: Ubuntu 22.04 with vulnerable e-commerce webapp (port 5002)
- **Local Testing**: Windows 11 with all services on localhost

B. Benchmark Methodology

We conducted large-scale testing with 150 payloads:

- 100 randomly generated SQL injection attacks using 20 attack templates
- 50 safe queries (normal search terms, emails, form inputs)
- 5 concurrent workers for realistic load
- 10 runs per payload for timing accuracy

C. Detection Results

Table III shows the full pipeline (Case 2) benchmark results.

TABLE IV
Benchmark Results - Full Pipeline (Case 2)

Detection Metric	Value	Performance	Value
True Positives	92	Avg Response	89ms
False Negatives	8	After Learning	$<$50ms
True Negatives	48	Requests/sec	5.6
False Positives	2	Rules Generated	33
Attack Detection	92.0%	Pattern Blocks	13%
False Positive Rate	4.0%	ML Blocks	87%
Precision	97.9%		
Recall	92.0%		
F1 Score	**94.8%**		

D. Response Time Analysis

A key finding is the significant speedup from pattern learning. We measured median response times for each scenario:

After the learning phase, attacks matching known patterns are blocked without invoking the ML model, reducing response time by 83% (from 89ms to 15ms).

TABLE V
RESPONSE TIME BY DEFENSE SCENARIO

Scenario	Median	Speedup
Firewall Pattern Reject	15ms	5.9×
Firewall Pass → ML Reject	89ms	1.0× (baseline)
Firewall Pass → ML Pass	85ms	1.0×

E. Learning Demonstration

The adaptive learning cycle was validated through sequential attack testing:

1) **Attack 1**: "' `UNION SELECT...`"
   - Firewall: No pattern match → Forward to ML
   - ML: Attack detected (confidence: 88%) → BLOCKED
   - Firewall: Add to collection → Extract patterns
2) **Attack 2**: Same payload repeated
   - Firewall: Pattern match! → BLOCKED immediately
   - ML: Not called (response time: 15ms vs 89ms)

F. Data Exfiltration Prevention

In passthrough mode (Case 0), the RL attack agent successfully extracts sensitive PII data including usernames, passwords, phone numbers, and addresses from the vulnerable webapp. In full pipeline mode (Case 2), all 5 data exfiltration attempts were blocked, preventing exposure of 5 user records.

VI. DISCUSSION

A. Strengths

- **Continuous Improvement**: Unlike static WAFs, the system learns from each attack
- **Layered Defense**: Multiple detection methods provide defense-in-depth
- **Transparency**: Generated Snort rules can be audited, exported, and integrated with existing IDS infrastructure
- **Graceful Degradation**: Automatic CNN→RandomForest fallback ensures availability

B. Limitations

- **Cold Start**: Initial ML-only phase is slower until patterns accumulate (minimum 5 payloads required for clustering)
- **False Positives**: Low detection threshold (0.2) may block legitimate complex queries containing SQL-like syntax
- **Overfitting Patterns**: Some learned patterns (e.g., "SELECT", "FROM") may be too generic

C. Future Work

- Replace Q-learning with Deep Q-Networks (DQN) for larger state spaces
- Extend pattern extraction to XSS and command injection
- Implement pattern generalization using regex to reduce overly specific rules
- Add confidence-weighted pattern matching

VII. CONCLUSION

This paper presented an integrated offensive-defensive AI system for SQL injection security. The hybrid RL-BiLSTM offensive agent achieves 99.5% WAF bypass rate for penetration testing. The dual-model ML detector provides 92% attack detection with automatic fallback. The adaptive firewall learns from detected attacks using DBSCAN clustering, generating 33 Snort rules during testing and reducing response time by 83% after pattern learning.

The modular architecture allows independent deployment while the feedback loop ensures continuous improvement. Our results demonstrate that combining offensive AI testing, ML-based detection, and unsupervised pattern learning creates a robust, self-improving defense system suitable for protecting web applications in production environments.

ACKNOWLEDGMENT

This work was developed as part of the ICS Security course at Technical University Gheorghe Asachi Iași.

REFERENCES

[1] OWASP, "OWASP Top 10 - 2021: A03 Injection," 2021. [Online]. Available: https://owasp.org/Top10/
[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.
[3] M. Liu, K. Li, and T. Chen, "DeepSQLi: Deep Semantic Learning for Testing SQL Injection," in *Proc. ACM SIGSOFT ISSTA*, 2020, pp. 286-297.
[4] B. D. Damele and M. Stampar, "SQLMap - Automatic SQL injection and database takeover tool," 2019.
[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD*, 1996, pp. 226-231.
[6] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proc. EMNLP*, 2014, pp. 1746-1751.
[7] Snort Team, "Snort Rules Writers Guide," Cisco Talos, 2023.