AI-Enhanced SQL Injection Detection Framework: A Novel Approach Combines LLMs with Traditional Fuzzing to Improve Web Application Vulnerability Detection

DOI: https://doi.org/10.54654/isj.v3i26.1179

Nguyen Le Quoc Dat*, Nguyen Le Quoc Anh, Nguyen Manh Thang

***Abstract*— SQL injection affects 65% of web applications, yet traditional tools often miss context-specific vulnerabilities. We propose AESIDF, a hybrid framework that integrates Large Language Models with parallel fuzzing for semantic vulnerability analysis. Evaluated on 26 benchmark scenarios from PortSwigger, DVWA, and OWASP Juice Shop, our approach achieves a 92.3% detection rate compared to SQLMap's 76.9%, while reducing request volume by approximately 68.8%. These preliminary results suggest that LLM-powered contextual reasoning can enhance automated security testing; however, broader validation on larger and more diverse datasets is required to confirm generalizability.**

***Tóm tắt*— Tấn công SQL injection ảnh hưởng đến khoảng 65% ứng dụng web, trong khi các công cụ truyền thống thường bỏ sót các lỗ hổng phụ thuộc ngữ cảnh. Chúng tôi đề xuất AESIDF(AI-Enhanced SQLi Detection Framework), một khung lai kết hợp Mô hình ngôn ngữ lớn (LLM) với kỹ thuật fuzzing song song nhằm phân tích lỗ hổng theo ngữ nghĩa. Được đánh giá trên 26 kịch bản benchmark từ PortSwigger, DVWA(Damn Vulnerable Web Application) và OWASP(Open Web Application Security Project) Juice Shop, phương pháp của chúng tôi đạt tỷ lệ phát hiện 92,3% so với 76,9% của SQLMap, đồng thời giảm khoảng 68,8% số lượng request. Các kết quả sơ bộ này cho thấy khả năng suy luận ngữ cảnh của LLM(Large Language Model) có thể cải thiện kiểm thử bảo mật tự động; tuy nhiên, cần xác thực rộng hơn trên các tập dữ liệu lớn và đa dạng hơn để khẳng định khả năng tổng quát hóa.**

Keywords*— *SQL Injection, artificial intelligence, web security, penetration testing, large language model, burp suite extension.

Từ khóa*— *SQL Injection, trí tuệ nhân tạo, an toàn web, kiểm thử xâm nhập, mô hình ngôn ngữ lớn; phần mở rộng burp suite.

I. INTRODUCTION

A. Motivation and research gap

SQL Injection (SQLi) consistently ranks as one of the most critical web security threats, appearing at the top of the OWASP Top 10 for over a decade [1] SQLi is considered a common type of attack on websites and web applications (hereinafter referred to as web attacks) [8] . Despite extensive research, 65% of modern applications remain vulnerable. Traditional detection tools face fundamental limitations: signature-based methods cannot cover all contexts, static patterns fail against novel encoding schemes, and the lack of contextual understanding causes them to miss application-specific vulnerabilities.

Research Gap: Existing methods fail to understand semantic context-they cannot distinguish between legitimate error messages and actual vulnerabilities, adapt to application-specific behaviors, or generate context-aware payloads. This semantic blindness leads to both false positives and missed vulnerabilities. The emergence of Large

This manuscript was received on October 31, 2025. It was reviewed on December 9, 2025, revised on December 18, 2025 and accepted on December 22, 2025.

* Corresponding author.

Language Models presents an unprecedented opportunity to bridge this gap through contextual reasoning and adaptive payload generation.

This work addresses three fundamental research questions:

- RQ1: How can LLMs reduce false positives in SQL injection detection through contextual analysis?
- RQ2: Can AI-driven payload generation detect advanced SQLi variants that evade traditional tools?
- RQ3: What is the optimal architecture for integrating LLMs into real-time security workflows?

B. Main contributions

- Hybrid Detection Framework: First systematic integration of LLM reasoning with parallel fuzzing for SQL injection detection
- Adaptive Payload Generation: Context-aware exploitation capable of bypassing modern WAF(Web Application Firewall) filters through semantic understanding
- Empirical Validation: Demonstrated 92.3% detection rate on 26 benchmark scenarios, with no observed false positives in this positive-only dataset (note: FPR(False Positive Rate) cannot be conclusively computed without negative samples)
- Open-Source Tool: Production-ready Burp Suite extension enabling seamless integration into existing penetration testing workflows

C. Paper organization

The remainder of this paper is organized as follows: Section II provides background on SQL injection fundamentals and detection challenges. Section III reviews related work and highlights research gaps. Section IV presents the problem statement. Section V describes our proposed method and architecture. Section VI reports the experimental evaluation including dataset, baselines, metrics, and key findings. Section VII discusses the strengths, limitations, and implications of our work. Section VIII outlines potential future work. Finally, Section IX concludes the paper.

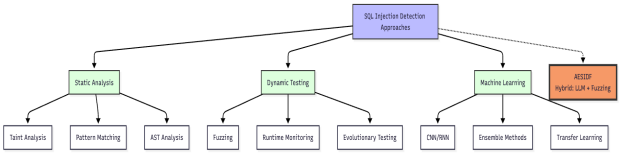


Figure 1. Taxonomy of SQLi detection approaches

II. Background

A. SQL Injection fundamentals

SQL injection exploits improper input sanitization in database queries. Attack vectors include:

- Error-based: Extracting data through database error messages
- Boolean-based blind: Inferring data through conditional responses
- Time-based blind: Using delays to extract information
- Out-of-band: Exfiltrating data through alternative channels
- UNION-based: Combining results from multiple SELECT statements
- Stacked queries: Executing multiple SQL statements in sequence

B. Detection challenges

Modern applications employ multiple defense layers:

- Web Application Firewalls (WAFs) with pattern matching
- Input encoding and parameterized queries
- Custom sanitization functions
- Rate limiting and behavioral analysis

Traditional detection tools struggle to bypass these defenses due to their reliance on static patterns and lack of contextual understanding. This motivates the need for more intelligent, adaptive approaches.

III. Related Work

A. Static analysis approaches

Taint Analysis: Saiyed et al. [2] track data flow from user inputs to SQL queries, achieving 78% accuracy. However, requires source code access and fails on obfuscated code.

Pattern Matching: Traditional scanners use regex patterns to identify vulnerable code constructs. Limited by inability to understand runtime behavior and context.

B. Dynamic testing methods

Runtime Monitoring: AMNESIA [3] intercepts queries at runtime, comparing against legitimate query models (85% accuracy). High deployment overhead limits practical adoption.

Fuzzing-based: SQLMap remains the industry standard, using extensive payload databases. Achieves 75% detection on our benchmark but requires manual configuration and misses context-specific vulnerabilities.

Evolutionary Testing: Appelt et al. [4] use genetic algorithms to evolve test cases (89% coverage). Computational cost prohibits real-time usage.

C. Machine learning approaches

Deep Learning: Tang et al. [5] apply CNNs to classify SQL queries (94.2% accuracy). Models trained on fixed datasets cannot adapt to novel attacks.

Ensemble Methods: Recent work combines multiple classifiers but still relies on static feature extraction, missing semantic context.

AI-based Exploitation: Thien et al. [9] use reinforcement learning for automated vulnerability exploitation with integrated scanning tools.

ML-enhanced WAF: Ho et al. [10] improve WAF performance using SVM with HTTP request analysis.

LLM-Integrated Fuzzing: Jiang et al. [7] survey LLM-enhanced fuzzing techniques, highlighting challenges in scalability and cost that motivate hybrid approaches.

D. Research gap

A review of existing approaches (Table I and Figure 1) highlights three major gaps:

- Lack of Semantic Reasoning: Current methods operate primarily on syntactic rules or statistical features, without understanding application logic.
- Poor Adaptability: Traditional tools cannot dynamically adapt payloads to diverse application contexts or bypass custom filters.
- Accuracy–Efficiency Trade-off: Existing solutions either achieve high detection at the cost of excessive requests, or prioritize speed but miss complex vulnerabilities.

TABLE I. COMPREHENSIVE COMPARISON OF DETECTION APPROACHES

Method	Technique	Acc.	FPR	Key Limitation
Static [2]	Taint	78%	22%	Source required
AMNESIA [3]	Runtime	85%	15%	High overhead
Evolutionary [4]	Genetic	89%	18%	Slow execution
CNN [5]	Deeplearning	94%	12*	Fixed patterns
SQLMap	Fuzzing	75%	**Low***	No context
AESIDF	**LLM+Fuzz**	**92.3%**	**Low***	**API cost**

Results are indicative only, as evaluation datasets differ across studies.

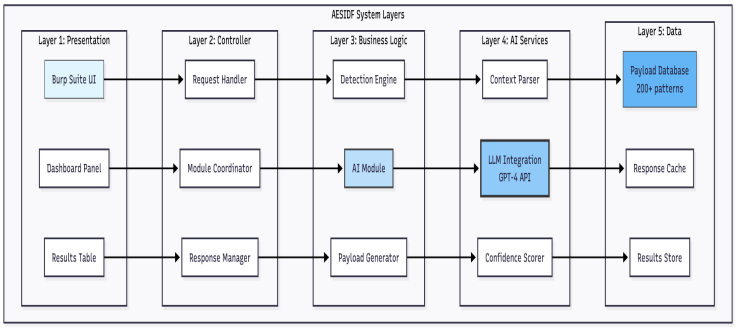


Figure 2. AESIDF system design: (a) five-layer modular architecture; (b) detection workflow across traditional and AI components

Our Contribution: AESIDF addresses these gaps by integrating LLM-powered semantic analysis with parallel fuzzing, enabling context-aware payload generation, robust false positive control, and efficient detection (92.3% accuracy with 68.8% fewer requests).

IV. PROBLEM STATEMENT

Despite decades of research, SQLi detection faces three fundamental limitations:

Context Blindness: Tools cannot understand application semantics, leading to inappropriate payloads and missed vulnerabilities.

Static Patterns: Fixed payload databases fail against novel encoding schemes and custom filters.

False Positives: Traditional fuzzing-based tools often misclassify benign errors as attacks, creating unnecessary alerts and increasing validation workload for security teams.

These limitations necessitate a paradigm shift from pattern matching to semantic understanding, motivating our LLM-enhanced approach.

V. Proposed Method

A. System architecture

AESIDF employs a five-layer modular architecture (Figure 2):

- Presentation Layer: BurpSuite UI for visualization and reporting.
- Controller Layer: Handles requests, coordinates modules, and manages responses.
- Business Logic Layer: Includes the detection engine, payload generator, and AI-assisted analysis.
- AI Services Layer: Provides contextual parsing, LLM integration, and confidence scoring.
- Data Layer: Stores payload databases, response cache, and results.

The detection pipeline operates in five stages: request interception, baseline fuzzing, AI-guided analysis, adaptive testing, and final reporting.

B. Runtime SQLi detection process

Figure 3 illustrates how AESIDF detects SQLi vulnerabilities during runtime execution. When the extension intercepts HTTP traffic through Burp Suite's proxy:

- Request Interception: Captures live HTTP requests containing potential injection points
- Parallel Fuzzing: Spawns multiple concurrent threads, each testing a different payload category.
- Response Analysis: Monitors for SQL errors, timing anomalies (>3s), content variations
- AI Evaluation: LLM analyzes suspicious patterns and generates context-aware payloads
- Confirmation: Validates vulnerabilities with targeted exploitation attempts

C. Detection pipeline

Our five-stage detection pipeline processes each request in real-time:

Stage 1 - Fuzzing: Parallel execution monitoring error patterns, response times (>2σ), content changes

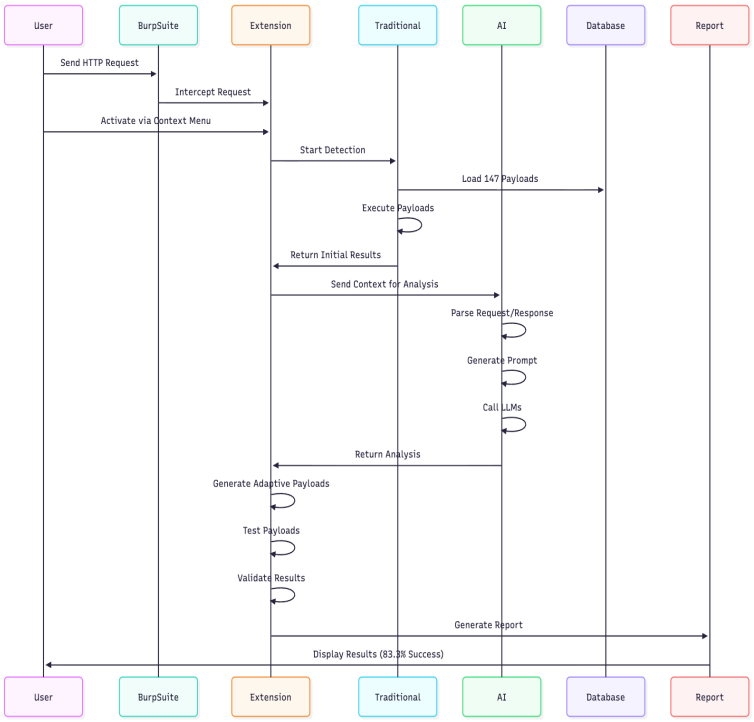


Figure3. Real-time SQL injection detection sequence showing how AESIDF intercepts HTTP traffic, executes parallel fuzzing, and applies AI analysis in runtime

Stage 2 - Fingerprinting: Database identification via concatenation operators, version functions, error signatures.

Stage 3 - AI Filtering: LLM semantic analysis with confidence scoring (0-100%).

Stage 4 - Adaptive Payloads: DBMS-specific generation (PostgreSQL ::int, MySQL BENCHMARK, Oracle XMLType, MS SQL WAITFOR).

Stage 5 - Confirmation: Final validation through targeted exploitation.

D. LLM integration

The AI module integrates either GPT-4(Generative Pre-trained Transformer-4) or locally deployed LLMs for contextual analysis, enabling:

- Confidence scoring (0–100%) for vulnerability assessment
- Context-aware payload generation
- WAF bypass strategy synthesis
- Semantic filtering to reduce false positives

Deployment considerations:

- GPT-4 API: Provides high accuracy and

advanced reasoning, but adds network latency, recurring costs, and potential privacy/compliance risks when sending data externally.
- Local LLMs: Improves privacy and eliminates API costs, but demands heavy compute resources, higher operational overhead, and often slower responses for large models.

System Prompt: You are an expert security analyst specializing in SQL injection vulnerability detection and exploitation. Your role is to analyze HTTP request/response pairs to determine if SQL injection vulnerabilities exist.

Core Capabilities:

- Identify SQL injection vulnerabilities with high accuracy
- Determine the database type and version
- Classify injection types (Union, Boolean, Time-based, Error-based, Stacked, Second-order)
- Detect WAF/security filters and suggest bypass techniques
- Generate effective payloads for further testing
- Assess exploitability and provide remediation guidance

E. Implementation details

AESIDF is implemented as a Burp Suite extension using Java 17 with the Montoya API and optional GPT-4 integration. The system leverages ExecutorService for parallel fuzzing with multiple threads and maintains multiple payload databases for different injection types and database systems.

Reproducibility specifications:

- LLM Configuration: Experiments used GPT-4 (gpt-4-turbo-preview) via OpenAI API with temperature=0.3 for deterministic outputs. Alternative local deployment tested with LLaMA-2 13B.
- Confidence Threshold: Vulnerability alerts are triggered when the LLM confidence score exceeds 70%. This threshold was empirically determined through preliminary testing on a validation subset, balancing detection sensitivity against false alerts.
- Timing Parameters: Time-based detection uses a 3-second delay threshold; responses exceeding 2σ from baseline are flagged for AI analysis.
- Thread Configuration: Parallel fuzzing uses 4 concurrent threads per endpoint to balance throughput and server load.

VI. Experiments and Results

A. Dataset

Primary benchmark: PortSwigger Web Security Academy. This dataset includes 18 SQL Injection labs, categorized as follows:

- 4 UNION-based attacks
- 6 Blind SQL Injection (conditional responses, conditional errors, time-based)
- 2 Out-of-band scenarios
- 6 Miscellaneous (login bypass, hidden data retrieval, DB version/query disclosure, database enumeration)

Additional testing:

- Damn Vulnerable Web Application (DVWA) – 3 levels (Low, Medium, High) for the SQL Injection module
- OWASP Juice Shop – 5 challenges related to SQL Injection

Total: 26 test cases (all positive/vulnerable scenarios).

Dataset justification and limitations: We acknowledge that this dataset consists entirely of lab-generated environments rather than real-world enterprise applications. This design choice was made for several reasons: (1) ethical and legal constraints prevent testing on production systems without explicit authorization; (2) PortSwigger, DVWA, and Juice Shop are widely recognized benchmarks in the security research community, enabling reproducibility and comparison with prior work; (3) these platforms provide controlled, well-documented vulnerability scenarios covering diverse SQLi attack vectors.

However, this approach has inherent limitations. The 26 test cases represent a statistically modest sample size, insufficient to make strong claims about generalizability. Furthermore, all scenarios are positive (vulnerable) cases-no negative (non-vulnerable) samples are included, which prevents computation of a meaningful False Positive Rate (FPR). Future work should expand the dataset to include

negative samples and real-world application clones to strengthen empirical validation.

B. Baselines

- SQLMap 1.7: Industry-standard fuzzing tool (Traditional fuzzing-based)
- AESIDF: AI-Enhanced SQL Injection Detection Framework – a novel approach that combines Large Language Models (LLMs) with traditional fuzzing to improve web application vulnerability detection

C. Evaluation metrics

We evaluate the approaches using four key metrics:

- Detection Rate (Recall): $TP/(TP + FN)$. This is our primary metric given the positive-only dataset.
- Observed False Positives: Since all 26 test cases are confirmed vulnerabilities (no negative samples), we cannot compute FPR = $FP/(FP + TN)$. Instead, we report the count of observed false positives—cases where the tool incorrectly flagged a vulnerability that did not exist or misclassified the vulnerability type.
- Detection Time: Estimated seconds per endpoint, computed as (average requests × average RTT). *Note: This estimation does not fully capture LLM inference latency and API round-trip overhead; actual end-to-end time may vary depending on network conditions and LLM provider load.*
- Resource Efficiency: Average HTTP/API requests per endpoint. Lower request counts indicate more efficient probing strategies.

D. Experimental results

As shown in Table II and Table III:

- AESIDF detected 24/26 vulnerabilities (92.3%) compared to SQLMap's 20/26 (76.9%) on this benchmark dataset.
- Both tools exhibited zero observed false positives; however, since all test cases are confirmed vulnerabilities (positive-only dataset), FPR cannot be meaningfully computed. This result should not be interpreted as evidence of low FPR in production environments.

TABLE II. SQL INJECTION DETECTION RESULTS (AESIDF VS SQLMAP)

Application / Lab	AESIDF	SQLMap
DVWA		
Low	✓	✓
Medium	✓	✓
High	✓	X
Juice Shop		
Task 2 (Error)	✓	X
Task 3 (Admin)	✓	✓
Task 4 (Jim)	✓	X
Task 5 (Bender)	✓	X
Task 10 (Search)	✓	✓
PortSwigger Labs		
Determine number of columns (UNION)	✓	✓
Find column containing text (UNION)	✓	✓
Retrieve data from other tables (UNION)	✓	✓
Retrieve multiple values in single column (UNION)	✓	✓
Query DB version (Oracle)	✓	✓
Query DB version (MySQL/MSSQL)	✓	✓
Listing DB contents (non-Oracle)	✓	✓
Listing DB contents (Oracle)	✓	✓
Retrieve hidden data (WHERE clause)	✓	✓
Login bypass	✓	✓
Blind SQLi – conditional responses	✓	✓
Blind SQLi – conditional errors	✓	✓
Blind SQLi – time delays	✓	✓
Blind SQLi – time delays + info retrieval	✓	✓
Blind SQLi – out-of-band interaction	X	X
Blind SQLi – out-of-band data exfiltration	X	X

TABLE III. EVALUATION METRICS COMPARISON (AESIDF VS SQLMAP)

Metric	AESIDF	SQLMap
Detection Rate (Recall)	92.3% (24/26)	76.9% (20/26)
False Positives (FP)	0	0
Avg. Requests / Endpoint	∼250	∼800
Detection Time / Endpoint	∼25 s	∼80 s

- AESIDF identified cases missed by SQLMap: DVWA-High (stricter input filtering), Juice Shop Tasks 2/4/5 (non-standard injection contexts), suggesting improved handling of WAF-protected scenarios.
- AESIDF consumed approximately 68.8% fewer requests (∼250 vs ∼800 per endpoint), indicating more efficient probing through semantic prioritization.
- Estimated detection time was approximately

68.8% shorter (∼25s vs ∼80s per endpoint). *Caveat: These estimates are derived from request count × average RTT and do not include LLM API latency overhead.*
- Both tools failed on out-of-band (OOB) scenarios, which require external DNS/HTTP callback infrastructure not configured in our test environment.

VII. Discussion

A. Strengths

On the 26-scenario benchmark, AESIDF achieved a 92.3% detection rate compared to SQLMap's 76.9%. While this represents a 15.4 percentage point improvement, we emphasize that these results are preliminary and require validation on larger, more diverse datasets. Key observed strengths include:

- Semantic Understanding: LLM-powered contextual analysis enabled detection of vulnerabilities in scenarios with stricter input filtering (DVWA-High) and non-standard injection contexts (Juice Shop) that SQLMap missed.
- Adaptive Payload Generation: Context-aware payload synthesis demonstrated capability to bypass certain WAF filters through semantic understanding rather than brute-force pattern enumeration.
- Resource Efficiency: Approximately 68.8% reduction in request volume compared to SQLMap, suggesting that semantic prioritization can reduce redundant probing.
- Practical Integration: Implementation as a Burp Suite extension enables integration into existing penetration testing workflows without significant toolchain changes.

B. Limitations

Dataset constraints: AESIDF was evaluated on only 26 test cases from lab-generated environments (PortSwigger, DVWA, Juice Shop). This sample size is statistically insufficient for robust scientific claims about generalizability. A 92.3% detection rate on 26 samples does not guarantee comparable performance across diverse real-world web applications with varying architectures, frameworks, and defense mechanisms.

Absence of negative samples: All test cases are confirmed vulnerabilities. Without non-vulnerable (negative) samples, we cannot compute a meaningful False Positive Rate (FPR). While zero false positives were observed, this claim
cannot be extended to production environments, where distinguishing true vulnerabilities from benign behaviors is critical. Future evaluation must incorporate negative test cases.

Baseline limitations: Our comparison is limited to SQLMap 1.7. A more comprehensive evaluation should include additional baselines such as Sqlninja, jSQL Injection, or commercial DAST tools to provide broader context for AESIDF's performance.

Runtime estimation: Detection time estimates (∼25s vs ∼80s) are derived from request count × average RTT. This methodology does not account for LLM API latency, inference time, rate limiting, or network variability. Actual end-to-end performance may differ significantly in real deployments.

Additional operational limitations include:

- API Costs: GPT-4 integration incurs recurring costs ($0.01–0.03 per 1K tokens), which may be prohibitive for large-scale assessments.
- Privacy/Compliance: Transmitting request/response data to external LLM providers raises GDPR, HIPAA, and other compliance concerns.
- Scalability: Performance on large-scale applications (1000+ endpoints) remains unvalidated.
- Out-of-band detection: Both tools failed on OOB scenarios, indicating a gap in callback-based vulnerability detection.

C. Implications

While our results are preliminary, they suggest promising directions for AI-augmented security testing:

- Research directions: The integration of LLM reasoning with traditional fuzzing opens questions about adversarial robustness, prompt injection risks in security tools, and optimal human-AI collaboration in penetration testing workflows.
- Practical potential: If validated on larger datasets, semantic prioritization could reduce manual triage effort in vulnerability

assessment, though the 70% reduction estimate requires further empirical validation.
- Limitations of current paradigm: Both AESIDF and traditional tools failed on out-of-band scenarios, indicating that fundamental detection gaps persist regardless of AI augmentation.

VIII. Future Work

A. Addressing current limitations (Priority)

- Dataset expansion: Extend evaluation to 100–200+ test cases, incorporating real-world application clones (WordPress, e-commerce platforms, custom enterprise applications) to improve statistical significance.
- Negative sample inclusion: Add non-vulnerable endpoints and benign error scenarios to enable meaningful FPR computation and validate false positive control claims.
- Additional baselines: Compare against Sqlninja, jSQL Injection, OWASP ZAP, and commercial DAST tools for broader context.
- End-to-end timing measurement: Implement actual latency instrumentation capturing LLM API calls, inference time, and network overhead rather than request-count estimates.

B. Technical enhancements

- Local LLMs: Deploy LLaMA-2 70B or Mistral for privacy preservation and cost reduction.
- Multi-Agent Architecture: Extend to XSS, CSRF, XXE detection using specialized agents.
- Adversarial Resilience: Train on evasion techniques and implement confidence calibration to handle adversarial inputs.
- Out-of-band support: Integrate callback infrastructure (DNS/HTTP) to address the OOB detection gap identified in our evaluation.

C. Application domains

- NoSQL Injection: MongoDB, GraphQL, JSON-based attacks.
- API Security: REST fuzzing, OAuth analysis, JWT vulnerabilities.
- Cloud Security: Serverless, containers, Kubernetes configurations.

D. Research directions

- Theoretical: Formal verification of AI-generated payloads and analysis of prompt injection risks in security tools.
- Empirical: Large-scale evaluation on 1000+ diverse applications with mixed positive/negative samples.
- Comparative: Human expert vs AI detection accuracy study to quantify the value of AI augmentation.

IX. Conclusion

Contributions: We introduced AESIDF, a framework that integrates Large Language Models with parallel fuzzing for SQL injection detection. The system provides: (1) semantic analysis of HTTP traffic for vulnerability assessment, (2) adaptive payload generation informed by application context, and (3) AI-assisted response interpretation to aid vulnerability confirmation.

Findings: On a benchmark of 26 scenarios from DVWA, Juice Shop, and PortSwigger Labs, AESIDF achieved a 92.3% detection rate (24/26) compared to SQLMap's 76.9% (20/26), with approximately 68.8% fewer requests per endpoint. Both tools exhibited zero observed false positives on this positive-only dataset; however, the absence of negative samples precludes reliable FPR estimation.

Limitations and caveats: These results are preliminary. The 26-case dataset is statistically modest and consists entirely of lab-generated environments. Generalizability to real-world applications with diverse architectures and defense mechanisms requires further validation. Claims about false positive control cannot be conclusively made without negative test cases.

Outlook: This work provides initial evidence that LLM-augmented approaches can enhance SQL injection detection efficiency and coverage in controlled settings. Future work should prioritize: (1) expanding evaluation to larger, more diverse datasets including negative samples; (2) benchmarking against additional baselines; and (3) measuring actual end-to-end performance including LLM overhead. We hope AESIDF serves as a foundation for further research on AI-augmented security testing.

ACKNOWNLEDGMENT

We would like to thank the Academy of Cryptography Technique for supporting this research. We also acknowledge PortSwigger, OWASP, and the security research community for providing the benchmark datasets and tools that made this evaluation possible.

REFERENCES

[1] OWASP Foundation, "OWASP Top 10 - 2021: The Ten Most Critical Web Application Security Risks" (2021). Access time: 12/01/2024, *https://owasp.org/Top10/*.

[2] S. Saiyed, H. Sharma, and V. Kumar, "Static analysis approach for SQL injection vulnerability detection," *Journal of Information Security and Applications*, vol. 45, pp. 1–12, 2019.

[3] W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and monitoring for neutralizing SQL-injection attacks," *ACM Transactions on Software Engineering*, vol. 34, no. 5, pp. 1–52, 2008.

[4] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for SQL injection vulnerabilities: An input mutation approach," in *Proc. Int. Symp. Software Testing and Analysis (ISSTA)*, San Jose, CA, USA, Jul. 2014, pp. 259–269.

[5] Y. Tang, S. Chen, and L. Wang, "SQL injection detection using convolutional neural networks," *IEEE Access*, vol. 8, pp. 132505–132517, 2020.

[6] T. Brown et al., "Language models are few-shot learners," in *Proc. NeurIPS*, 2020, pp. 1877–1901.

[7] Y. Jiang et al., "When Fuzzing Meets LLMs: Challenges and Opportunities," in *Proc. ACM CCS*, Salt Lake City, USA, 2024, pp. 1–15.

[8] H. X. Dau, N. T. T. Trang, and N. T. Hung, "A Survey of Tools and Techniques for Web Attack Detection," *Journal of Science and Technology on Information Security*, Special Issue CS, no. 15, pp. 109–130, 2022. DOI: doi.org/10.54654/isj.v1iCS(15).403.

[9] N. M. Thien, P. D. Khoa, N. D. Vuong, and N. V. Hung, "AI application framework for automatic vulnerabilities exploit," *Journal of Science and Technology on Information Security*, vol. 1, no. 13, pp. 80–92, 2022. DOI: doi.org/10.54654/isj.v1i13.234.

[10] T. P. Ho, H. T. Nam, and N. M. Thang, "A new approach to improving web application firewall performance based on support vector machine method with analysis of HTTP request," *Journal of Science and Technology on Information Security*, vol. 1, no. 15, pp. 62–73, 2022. DOI: doi.org/10.54654/isj.v1i15.391.

ABOUT THE AUTHORS



Nguyen Le Quoc Dat

Workplace: CyEyes company, Vietnam

Email: datnguyenlequoc2001@gmail.com

Education: Engineer of Information Security, University of Information Technology - HCMVNU, Currently pursuing a masterâ€™s degree at Academy of Cryptography Technique.

Recent research interests: Information security, penetration testing automation, large language models for cybersecurity.

Tên tác giả: **Nguyễn Lê Quốc Đạt**

Email: datnguyenlequoc2001@gmail.com

Quá trình đào tạo: Kỹ sư An toàn thông tin, Đại học Công nghệ Thông tin - Đại học Quốc gia Hồ Chí Minh

Hướng nghiên cứu hiện nay: An toàn thông tin, tự động hóa kiểm thử xâm nhập, mô hình ngôn ngữ lớn cho an ninh mạng.



Nguyen Le Quoc Anh

Workplace: CyEyes company, Vietnam

Email: haniz.cons@gmail.com

Education:Bachelor of Information Security, Academy of Cryptography Technique, Currently pursuing a master's degree at Academy of Cryptography Technique.

Recent research interests: Information Security, Cybersecurity, Web Application Security, Penetration Testing.

Tên tác giả: **Nguyễn Lê Quốc Anh**

Cơ quan công tác: Công ty CyEyes, Việt Nam

Email: haniz.cons@gmail.com

Quá trình đào tạo: Cử nhân An toàn thông tin, Học viện Kỹ thuật Mật mã.

Hướng nghiên cứu hiện nay: An toàn thông tin, an ninh mạng, an toàn ứng dụng web, kiểm thử xâm nhập.



Nguyen Manh Thang

Workplace: Academy of Cryptography Technique, Vietnam Government Information Security Commission

Email: chieumatxcova@hotmail.com

Education: PhD of Information Security, Academy of Cryptography Technique.

Recent research interests: Information Security, Penetration Testing, Web Application Security, Security Automation.

Tên tác giả: **Nguyễn Mạnh Thắng**

Cơ quan công tác: Học viện Kỹ thuật Mật mã, Ban Cơ yếu Chính phủ, Việt Nam

Email: chieumatxcova@hotmail.com

Quá trình đào tạo: Tiến sĩ An toàn thông tin, Học viện Kỹ thuật Mật mã.

Hướng nghiên cứu hiện nay: An toàn thông tin, Kiểm thử xâm nhập, An toàn ứng dụng web, Tự động hóa bảo mật.