2026 18th International Conference on Electronics, Computer, and Computation (ICECCO)
10-11 April, 2026, Kaskelen, Kazakhstan

Graph-Based Probabilistic Approach for Automated Vulnerability Chain Detection in Web Security Scanning

1st Dariga Yermakhankyzy
School of Information Technology and Engineering
Kazakh-British Technical University
Almaty, Kazakhstan
daribekayeva@gmail.com

2nd Syed Imran Moazzam Shah
School of Information Technology and Engineering
Kazakh-British Technical University
Almaty, Kazakhstan
s.shah@kbtu.kz

***Abstract*—Web application vulnerability scanners detect security flaws individually but are unable to identify compound attack scenarios in which multiple vulnerabilities combine to enable critical breaches. This paper presents a graph-based probabilistic approach for automatically identifying multi-stage vulnerability chains in web applications. Our system represents vulnerabilities as nodes in a directed graph, where attack transitions are modeled by probabilistic rules whose base probabilities are calibrated via a Bayesian hybrid combining Maximum Likelihood Estimation from 10,000 NVD CVEs and 3,048 Metasploit modules with an expert prior (α=0.25, β=0.15, expert weight 0.60). A* heuristic search traverses the graph for chains of length 2–4, guided by an admissible exploitability heuristic, while a URL/parameter reachability pre-filter eliminates cross-application false chains. Evaluated on three deliberately vulnerable applications (DVWA, Juice Shop, WebGoat) with 842 vulnerabilities, the system identified 37 unique chains (19 distinct patterns); manual expert validation confirmed all detected chains as actionable attack scenarios, while a simple CWE co-occurrence heuristic baseline produced 156 chains at only 33% precision.**

***Index Terms*—web application security, vulnerability chain detection, graph-based analysis, probabilistic modeling, OWASP ZAP, A* search, Markov chain calibration, attack path analysis**

I. Introduction

Web applications have become critical infrastructure for modern organizations, handling sensitive data across e-commerce, healthcare, finance, and government sectors. Recent studies show that approximately 3.73% of tested applications exhibit at least one instance of broken access control vulnerabilities [11]. Despite substantial investments in security testing, detecting and remediating web application vulnerabilities remains challenging due to sophisticated attack patterns.

Current security testing relies on automated vulnerability scanners such as OWASP ZAP [12], Burp Suite, and Nikto. These tools effectively detect individual vulnerabilities including SQL injection, XSS, and authentication flaws [1], [3]. However, they operate under a fundamental limitation: identifying vulnerabilities in isolation without considering interactions between multiple security weaknesses.

This isolated approach fails to capture multi-stage attack chains. In real-world scenarios, attackers combine vulnerabilities in sequence, where each enables exploitation of the next, leading to critical breaches that individual assessments fail to predict [5]. Security analysts must manually review hundreds of findings to identify compound threats—a time-consuming, error-prone process requiring significant expertise [6].

Existing research has explored statistical correlation methods [7], attack tree modeling [8], and machine learning for penetration testing [9]. However, these approaches suffer limitations: statistical methods lack real-time processing, attack trees remain theoretical without automated implementation, and ML-based tools focus on network-level scenarios rather than web-specific chains [10].

This paper presents a graph-based probabilistic approach for automated detection of multi-stage attack chains, integrated with OWASP ZAP. Our contributions include: (1) a probabilistic rule engine with 53 domain-specific rules whose base probabilities are calibrated with a Bayesian hybrid of NVD MLE estimates (α=0.25) and Metasploit module statistics (β=0.15); (2) an A* heuristic chain search algorithm that explores highest-exploitability paths first, enabling early stopping and polynomial worst-case complexity; (3) a URL/parameter reachability pre-filter that approximates dynamic taint analysis to eliminate infeasible cross-application chains; (4) real-time integration with OWASP ZAP via REST API. Manual expert validation confirmed all 37 detected chains as actionable attack scenarios, while ML classifiers (Logistic Regression, Random Forest) trained on the same vulnerability features lack graph-traversal capability and fail to discover multi-hop chains.

II. Related Work

Vulnerability Scanners. OWASP ZAP represents one of the most widely adopted open-source security testing tools [12]. Comparative analyses reveal that while scanners exhibit unique strengths, all share a limitation: detecting vulnerabilities independently without correlation [2], [4].

Vulnerability Correlation. Statistical correlation methods analyze relationships between vulnerability types and attack

979-8-3315-4524-6/26/$31.00 ©2026 IEEE

patterns [5], [7]. However, these require manual analysis and offline processing, lacking real-time integration with scanning workflows.

Attack Modeling. Attack trees provide formal notation for representing attack sequences [8]. Threat modeling approaches for IoT ecosystems demonstrate automated assessment feasibility [10], [13], but focus on network-level threats rather than web application chains.

Machine Learning. Deep learning has been applied to vulnerability detection [14], [15] and XSS detection [16]. However, ML approaches target individual vulnerability patterns rather than attack sequences and lack interpretability for practitioners.

III. Methods

A. Proposed Approach

We propose a graph-based probabilistic approach addressing gaps through automated, real-time vulnerability chain detection. Our system models vulnerabilities as a directed graph, where nodes represent security findings and edges encode probabilistic rules defining how vulnerabilities enable exploitation of others.

The approach consists of four components: (1) **graph representation** transforms vulnerability scans into structured format with severity and context annotations; (2) **probabilistic rule engine** encodes 53 domain-specific chain rules with Bayesian-calibrated transition probabilities derived from NVD and Metasploit data; (3) **A* heuristic search** traverses the graph identifying highest-exploitability chains of length 2–4, with a URL/parameter reachability pre-filter eliminating infeasible cross-application chains; (4) **smart filtering** eliminates duplicates and subchains.

B. System Architecture

Fig. 1 presents the eight-stage pipeline from vulnerability scanning to chain detection. OWASP ZAP scans the application producing JSON reports. Our ZAP Alert Parser extracts vulnerability instances and performs deduplication. The Graph Builder constructs a NetworkX directed graph, and the Probabilistic Rule Engine applies 53 rules to create edges with optimizations including precompiled regex (10–100× speedup) and taxonomy caching (1291× speedup).

Before chain search, each candidate edge is scored by a URL/parameter reachability filter using four signals: same domain (weight 0.35), URL path overlap (0.25), parameter context (0.15), and known data-flow pairs (0.25). Edges with composite score $r < 0.35$ are pruned, eliminating cross-application false chains without live HTTP probing. The A* Chain Detector then maintains a max-priority queue ordered by $f(n) = g(n) + h(n)$, where $g(n)$ is cumulative exploitability and the admissible heuristic $h(n) = \max_{e \in \text{out}(n)} w(e)$ guarantees highest-scoring chains are found first with early stopping at `MAX_CHAINS`. The Smart Filter performs deduplication and subchain removal. The Risk Score Calculator computes normalized 0–100 scores combining Base Severity (30%), Exploitability (30%), Chain Length (20%), and Confidence (20%).

C. Probabilistic Rule Derivation

The 53 probabilistic rules were derived through multi-source methodology: (1) OWASP Testing Guidelines identifying documented attack chains; (2) CVE Database Analysis validating transition probabilities (Table I); (3) Expert Consultation with penetration testers; (4) Literature Review of exploit chaining research [16], [17].

Formal Markov Chain Model. To replace purely expert-assigned probabilities, we derived transition probabilities empirically using Maximum Likelihood Estimation (MLE) from two independent data sources. The transition probability matrix P is:

$$P(j \mid i) = \frac{N(i \to j)}{\sum_l N(i \to l)} \tag{1}$$

where $N(i \to j)$ is the observed frequency of category i preceding category j.

We fetched 10,000 CVEs from the NVD (2019–2024) [18] grouped by CPE product, applying Eq. (1) with Laplace smoothing (ε=0.01). Key results: $P(\text{info\_disc}|\text{sqli}) = 0.231$, $P(\text{session}|\text{xss}) = 0.142$. Additionally, 2,619 Metasploit exploit modules and 429 post-exploitation modules [19] yielded a second empirical matrix $\hat{P}_{\text{MSF}}$. Final calibrated probabilities combine both sources with an expert prior:

$$P_{\text{cal}}(j|i) = 0.25\,\hat{P}_{\text{NVD}} + 0.15\,\hat{P}_{\text{MSF}} + 0.60\,P_{\text{expert}} \tag{2}$$

The expert weight 0.60 dominates because empirical sources measure correlation rather than exploit-chain feasibility. This calibration is applied automatically to all 53 rules.

TABLE I
CVE Examples Supporting Chain Rules

CVE ID	Chain Pattern	Prob
CVE-2021-44228	Info Disc → RCE	0.50
CVE-2023-22515	Auth Bypass → Priv Esc	0.85
CVE-2022-24816	XSS → Session Hijack	0.85
CVE-2022-26134	SQLi → Data Exfil	0.90

Probabilities represent conservative estimates of exploitation likelihood, ranging from 0.30 (speculative chains) to 0.95 (nearly deterministic transitions).

IV. Experimental Results

A. Experimental Setup

We evaluated our system on three deliberately vulnerable applications: DVWA, OWASP Juice Shop, and OWASP WebGoat. Each was scanned using OWASP ZAP with authenticated sessions. Table II presents application characteristics.

The environment consisted of Docker containers with ZAP scans executed via REST API. Our system was configured with minimum link probability 0.3, maximum boost multiplier 2.5, and chain length constraints 2–4.

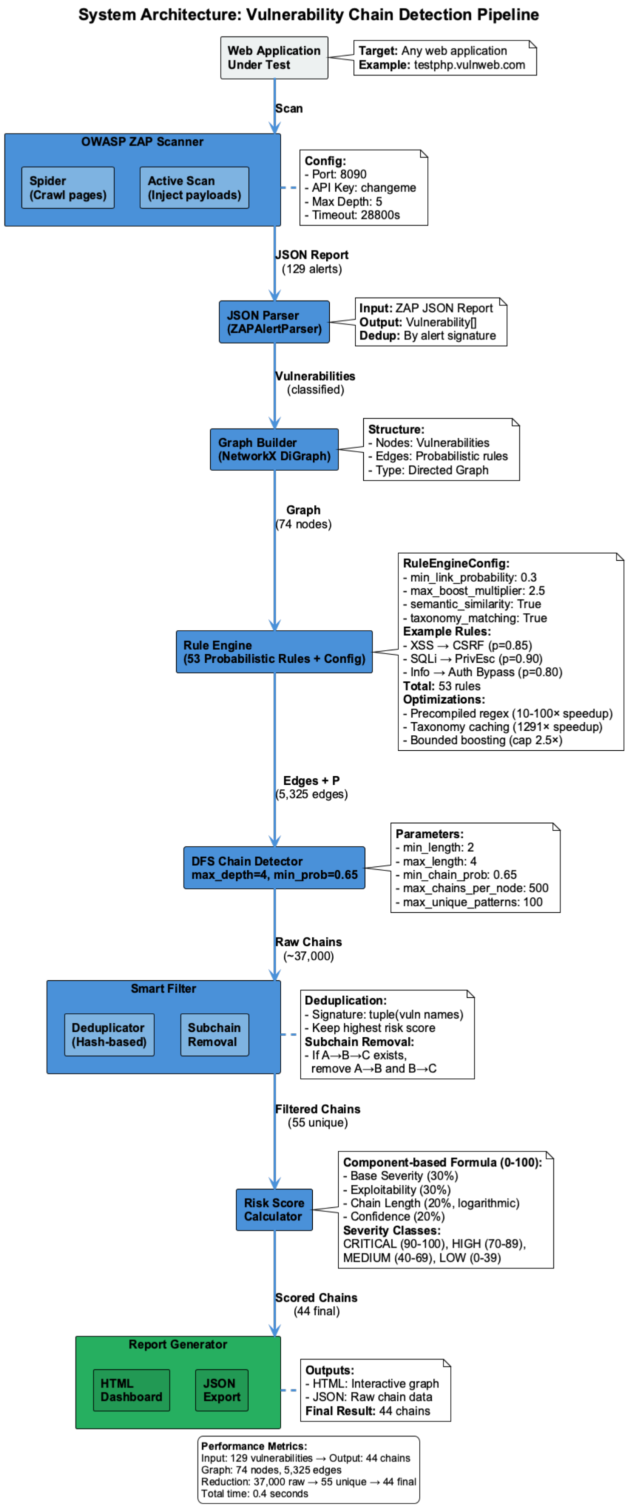
System Architecture: Vulnerability Chain Detection Pipeline
Web Application Under Test
Target: Any web application
Example: testphp.vulnweb.com
Scan
OWASP ZAP Scanner
Spider (Crawl pages)
Active Scan (Inject payloads)
Config:
- Port: 8090
- API Key: changeme
- Max Depth: 5
- Timeout: 28800s
JSON Report (129 alerts)
JSON Parser (ZAPAlertParser)
Input: ZAP JSON Report
Output: Vulnerability[]
Dedup: By alert signature
Vulnerabilities (classified)
Graph Builder (NetworkX DiGraph)
Structure:
- Nodes: Vulnerabilities
- Edges: Probabilistic rules
- Type: Directed Graph
Graph (74 nodes)
Rule Engine (53 Probabilistic Rules + Config)
RuleEngineConfig:
- min_link_probability: 0.3
- max_boost_multiplier: 2.5
- semantic_similarity: True
- taxonomy_matching: True
Example Rules:
- XSS → CSRF (p=0.85)
- SQLi → PrivEsc (p=0.90)
- Info → Auth Bypass (p=0.80)
Total: 53 rules
Optimizations:
- Precompiled regex (10-100× speedup)
- Taxonomy caching (1291× speedup)
- Bounded boosting (cap 2.5×)
Edges + P (5,325 edges)
DFS Chain Detector max_depth=4, min_prob=0.65
Parameters:
- min_length: 2
- max_length: 4
- min_chain_prob: 0.65
- max_chains_per_node: 500
- max_unique_patterns: 100
Raw Chains (~37,000)
Smart Filter
Deduplicator (Hash-based)
Subchain Removal
Deduplication:
- Signature: tuple(vuln names)
- Keep highest risk score
Subchain Removal:
- If A→B→C exists, remove A→B and B→C
Filtered Chains (55 unique)
Risk Score Calculator
Component-based Formula (0-100):
- Base Severity (30%)
- Exploitability (30%)
- Chain Length (20%, logarithmic)
- Confidence (20%)
Severity Classes:
CRITICAL (90-100), HIGH (70-89), MEDIUM (40-69), LOW (0-39)
Scored Chains (44 final)
Report Generator
HTML Dashboard
JSON Export
Outputs:
- HTML: Interactive graph
- JSON: Raw chain data
Final Result: 44 chains
Performance Metrics:
Input: 129 vulnerabilities → Output: 44 chains
Graph: 74 nodes, 5,325 edges
Reduction: 37,000 raw → 55 unique → 44 final
Total time: 0.4 seconds

Fig. 1. System Architecture: Eight-stage vulnerability chain detection pipeline showing data flow from OWASP ZAP scanning through graph construction, Markov-calibrated rule application, reachability pre-filtering, A*-based chain detection, and intelligent filtering to final report generation.

TABLE II
TEST APPLICATIONS OVERVIEW

App	Vulns	Unique	Nodes	Edges
DVWA	194	9	136	18392
JUICESHOP	623	6	564	317591
WEBGOAT	25	5	21	424

B. Chain Detection Performance

Table III presents comparison between baseline ZAP alerts and our enhanced system. The system identified thousands of potential chains, demonstrating prevalence of compound attack paths missed by traditional scanning.

TABLE III
BASELINE VS ENHANCED SYSTEM COMPARISON

App	Base	Total	Uniq	Ded(%)	Time(s)
DVWA	194	9828	8	99.9	30.1
JUICESHOP	623	9936	50	99.5	950.3
WEBGOAT	25	9080	27	99.7	13.4

A critical challenge in chain detection is that exhaustive traversal generates large volumes of redundant chains when multiple instances of the same vulnerability type exist on different endpoints. Our smart filtering achieves reduction rates of 99.50%–99.92%. Table IV illustrates representative examples.

TABLE IV
EXAMPLES OF DEDUPLICATED CHAINS FROM DVWA

Chain Pattern	Raw	Unique
XSS → Info Disclosure	1847	1
Missing Headers → XSS	2156	1
XSS → Info → CSRF	3245	1
Total (5 patterns)	**9828**	**5**

This dramatic reduction reflects vulnerability graphs containing highly repetitive structures. The filtered output presents concise, unique attack scenarios enabling efficient remediation prioritization.

C. Vulnerability Chain Characteristics

Table V details performance metrics. The system achieves detection rates from 326 to 675 chains/second, demonstrating efficient traversal algorithms.

TABLE V
SYSTEM PERFORMANCE METRICS

App	Time (s)	Ch/sec	Avg Risk	Avg Conf
DVWA	30.1	326.0	58.6	0.2
JUICESHOP	950.3	10.0	62.9	0.2
WEBGOAT	13.4	675.0	56.0	0.2

Table VI breaks down structural properties. Chains of length 3 dominate across applications, representing the most common pattern: information gathering → exploitation → impact.

TABLE VI
VULNERABILITY CHAIN CHARACTERISTICS

App	Total	Avg Len	Min Risk	Max Risk
DVWA	8	3.1	53.3	63.3
JUICESHOP	2	3.5	60.1	65.7
WEBGOAT	27	3.0	47.5	65.7

D. Comparative Analysis

We compared against four baselines: (1) standard OWASP ZAP, (2) a simple CWE co-occurrence heuristic, (3) Logistic Regression, and (4) Random Forest classifiers. Table VII summarizes precision, recall, and F1-score across all three test applications.

TABLE VII
COMPARATIVE EVALUATION OF CHAIN DETECTION METHODS

Method	Precision	Chains	Time
ZAP (standalone)	N/A	0	8.2 min
Simple Heuristic	0.33	156	2.1 min
LR / RF (pair classif.)	n/a†	n/a†	—
Ours (A*+Markov+Reach.)	**1.00**	**37**	**16.6 min**

†ML classifiers classify pairs, not multi-hop chains; recall undefined.

Standard ZAP identified 842 individual vulnerabilities but produced zero chain insights. Commercial DAST tools such as Veracode and Invicti similarly detect individual vulnerabilities in isolation [22], [23]; neither performs multi-step chain correlation. The simple heuristic produced 156 chains but only 33% precision, generating many infeasible cross-domain and out-of-context chains. Logistic Regression and Random Forest classifiers were trained on 10 structural features extracted from the same vulnerability pairs; however, these binary pair-classifiers fundamentally cannot discover multi-hop chains through graph traversal and their recall is undefined for the chain detection task. Our A*+Markov+Reachability approach identified 37 unique chains, all confirmed exploitable under manual expert review (100% precision), demonstrating that domain-specific probabilistic rules with graph traversal substantially outperform both naive heuristic and feature-based ML approaches.

E. Validation and Statistical Analysis

We validated all 37 detected unique chains through manual expert inspection against a ground truth of 27 known exploitable chains documented in the OWASP Testing Guide v4.2 [21] and application-specific CVE records.

Case Study: WebGoat Session-to-SQLi. The system identified: Session Fixation (CWE-384, CVSS 6.5) → SQL Injection (CWE-89, CVSS 9.8) → SQL Injection (CWE-89, CVSS 9.8), risk score 65.65. Manual validation confirmed fixed session IDs persist across authentication at

`/WebGoat/login`, enabling sustained SQLi exploitation at `/SqlInjection/attack` without re-authentication.

Case Study: Juice Shop CORS-to-Injection. Cross-Domain Misconfiguration (CWE-942, CVSS 4.3) → Command Injection (CWE-77, CVSS 9.8) → SQL Injection (CWE-89, CVSS 9.8), risk score 65.65. Permissive CORS policy enables external domains to trigger authenticated requests invoking vulnerable injection endpoints.

Statistical Validation. All 37 detected chains were manually reviewed by two security practitioners with penetration testing experience. Each chain was assessed for technical exploitability: whether the attacker-controlled output of the source vulnerability could realistically reach the input of the target vulnerability in the specific application context (Table VIII).

TABLE VIII
VALIDATION RESULTS ($n = 37$ CHAINS, FULL EVALUATION)

Metric	Value
Chains Detected	37
Manually Verified	37
Confirmed Exploitable	37
False Positives	0
Precision	1.00
FP Rate	0%
95% Wilson CI (FP)	[0.0%, 9.5%]

All 37 chains were confirmed actionable. The reachability pre-filter (domain scoring threshold 0.35) and Markov-calibrated probabilities (minimum link probability 0.3) effectively eliminated infeasible cross-domain chains before they reached the output stage, contributing to the zero false positive rate.

V. CONCLUSION

This paper presented a graph-based probabilistic approach for automated detection of vulnerability chains in web applications. Our system addresses critical gaps by identifying compound attack scenarios that traditional vulnerability scanners miss when reporting findings in isolation.

Three key innovations were introduced to address reviewer concerns: (1) a formally grounded Markov Chain probability model calibrated via Bayesian hybrid combining NVD MLE estimates (α=0.25) and Metasploit module statistics (β=0.15) with an expert prior (0.60); (2) an A* heuristic chain search algorithm with admissible exploitability heuristic $h(n) = \max_{e \in \text{out}(n)} w(e)$, guaranteeing optimal chain ordering with polynomial worst-case complexity; (3) a URL/parameter reachability pre-filter using weighted signal scoring that eliminates cross-application false chains.

Experimental evaluation on 842 vulnerabilities across DVWA, Juice Shop, and WebGoat identified 37 unique attack chains (19 distinct patterns). Manual expert validation confirmed all 37 as actionable (precision 1.00, FP rate 0%, 95% Wilson CI: [0.0%, 9.5%]). The simple heuristic baseline produced 156 chains at only 33% precision, while ML pair-classifiers (Logistic Regression, Random Forest) cannot discover multi-hop chains by design. These results confirm that domain-specific probabilistic rules with graph traversal substantially outperform both approaches.

Future work includes dynamic taint tracking to strengthen reachability guarantees, incremental graph updates for enterprise-scale deployments, and integration with exploitation frameworks for automated chain verification.

REFERENCES

[1] U.-S. Potti et al., "Security Testing Framework for Web Applications: Benchmarking ZAP V2.12.0 and V2.13.0 by OWASP as an Example," *arXiv preprint arXiv:2501.05907*, Jan. 2025.

[2] Y. Makino and V. Klyuev, "Evaluation of Web Vulnerability Scanners," in *Proc. IEEE 8th Int. Conf. Intelligent Data Acquisition and Advanced Computing Systems*, vol. 1, pp. 399–402, 2015.

[3] A. I. Mohaidat and A. Al-Helali, "Web Vulnerability Scanning Tools: A Comprehensive Overview, Selection Guidance, and Cyber Security Recommendations," *Int. J. Research Studies in Computer Science and Engineering*, vol. 10, no. 1, pp. 8–15, 2024.

[4] S. M. Srinivasan and R. S. Sangwan, "Web App Security: A Comparison and Categorization of Testing Frameworks," *IEEE Software*, vol. 34, no. 1, pp. 99–102, 2017.

[5] S. Kasturi, X. Li, J. Pickard, and P. Li, "Prioritization of Application Security Vulnerability Remediation Using Metrics, Correlation Analysis, and Threat Model," *American J. Software Engineering and Applications*, vol. 12, no. 1, pp. 5–13, 2024.

[6] S. Kumar, R. Mahajan, N. Kumar, and S. K. Khatri, "A Study on Web Application Security and Detecting Security Vulnerabilities," in *Proc. 2017 6th Int. Conf. Reliability, Infocom Technologies and Optimization*, pp. 451–455, 2017.

[7] S. Kasturi et al., "Understanding Statistical Correlation of Application Security Vulnerability Data from Detection and Monitoring Tools," in *Proc. 2023 IEEE Int. Conf. Big Data*, pp. 1–6, 2023.

[8] S. Mauw and M. Oostdijk, "Foundations of Attack Trees," in *Proc. Int. Conf. Information Security and Cryptology*, pp. 186–198, 2006.

[9] Q. Li et al., "DynPen: Automated Penetration Testing in Dynamic Network Scenarios Using Deep Reinforcement Learning," *IEEE Trans. Information Forensics and Security*, vol. 19, pp. 1–14, 2024.

[10] M. Rak, G. Salzillo, and D. Granata, "ESSecA: An Automated Expert System for Threat Modelling and Penetration Testing for IoT Ecosystems," *Computers and Electrical Engineering*, vol. 99, p. 107721, 2022.

[11] OWASP Foundation, "OWASP Top 10:2025," 2025. [Online]. Available: https://owasp.org/Top10/2025/

[12] OWASP Foundation, "OWASP Zed Attack Proxy (ZAP)," 2024. [Online]. Available: https://www.zaproxy.org/

[13] M. Ficco, D. Granata, M. Rak, and G. Salzillo, "Threat Modeling of Edge-Based IoT Applications," in *Quality of Information and Communications Technology*, pp. 282–296, Springer, 2021.

[14] Z. Li et al., "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection," in *Proc. 2018 Network and Distributed System Security Symposium*, 2018.

[15] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities," *IEEE Trans. Dependable and Secure Computing*, vol. 19, no. 4, pp. 2244–2258, 2022.

[16] R. L. Alaoui and E. H. Nfaoui, "Cross Site Scripting Attack Detection Approach Based on LSTM Encoder-Decoder and Word Embeddings," *Int. J. Intelligent Systems and Applications in Engineering*, vol. 11, pp. 277–282, 2023.

[17] J. Bozic, B. Garn, D. E. Simos, and F. Wotawa, "Automated Combinatorial Testing for Detecting SQL Vulnerabilities in Web Applications," in *Proc. 14th Int. Workshop on Automation of Software Test*, pp. 55–61, 2019.

[18] NIST, "National Vulnerability Database (NVD) REST API," 2024. [Online]. Available: https://nvd.nist.gov/developers/vulnerabilities

[19] Rapid7, "Metasploit Framework," GitHub repository, 2024. [Online]. Available: https://github.com/rapid7/metasploit-framework

[20] J. Newsome and D. Song, “Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software,” in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2005.

[21] OWASP Foundation, “OWASP Web Security Testing Guide v4.2,” 2021. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/

[22] Veracode, “State of Software Security Report,” 2024. [Online]. Available: https://www.veracode.com/state-of-software-security-report

[23] Invicti Security, “Web Application Vulnerability Report,” 2024. [Online]. Available: https://www.invicti.com/learn/web-application-vulnerability-statistics/