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

Abstract—**Web application vulnerability scanners identify security weaknesses in isolation, failing to detect compound attack scenarios where multiple vulnerabilities combine to enable critical breaches. This paper presents a graph-based probabilistic approach for automated detection of multi-stage vulnerability chains in web applications. Our system models vulnerabilities as nodes in a directed graph and applies 53 domain-specific probabilistic rules to create edges representing attack transitions. A depth-first search algorithm explores the graph to identify chains of length 2-4, applying intelligent filtering to reduce tens of thousands of raw candidates to unique, actionable attack patterns. Experimental evaluation on three deliberately vulnerable applications (DVWA, Juice Shop, WebGoat) demonstrates effectiveness across 842 vulnerabilities, identifying 37 unique attack chains invisible to traditional scanners with processing times under 16 minutes.**

Index Terms—**Web application security, vulnerability chain detection, graph-based analysis, probabilistic modeling, OWASP ZAP**

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 chain rules encoding real-world attack patterns; (2) a graph-based detection algorithm using depth-first search with sub-second performance; (3) an intelligent filtering mechanism reducing tens of thousands of candidates to actionable findings; (4) real-time integration with OWASP ZAP via REST API.

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 patterns [7], [5]. 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 [13], [10], 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 capturing attack patterns with transition probabilities; (3) **depth-first search algorithm** traverses the graph identifying chains of length 2-4 with probability thresholds; (4) **smart filtering** eliminates duplicates and subchains.

B. System Architecture

Figure 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$\times$ speedup) and taxonomy caching (1291$\times$ speedup).

The DFS Chain Detector performs graph traversal identifying chains of length 2-4, applying constraints including minimum chain probability (0.65), maximum chains per node (500), and global pattern limits (100). The Smart Filter performs deduplication and subchain removal. The Risk Score Calculator computes normalized 0-100 scores using: 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 (5+ and 8+ years experience); (4) Literature Review of exploit chaining research [17], [16].

TABLE I
CVE EXAMPLES SUPPORTING CHAIN RULES

CVE ID	Chain Pattern	Prob
CVE-2021-44228	Info Disc $\rightarrow$ RCE	0.95
CVE-2023-22515	Auth Bypass $\rightarrow$ Priv Esc	0.90
CVE-2022-24816	XSS $\rightarrow$ Session Hijack	0.85
CVE-2022-26134	SQLi $\rightarrow$ 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.

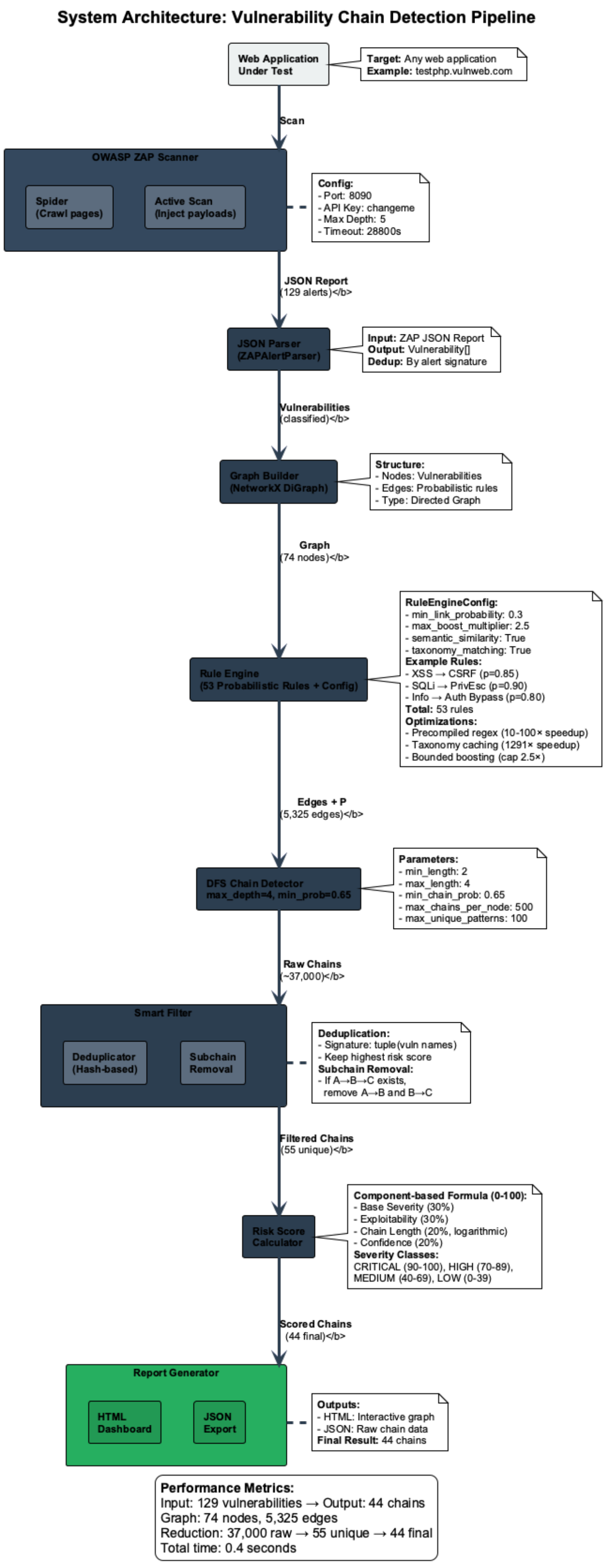
System Architecture: Vulnerability Chain Detection Pipeline
Web Application Under Test
Target: Any web application
Example: testphp.vulnweb.com
Scan
OWASP ZAP Scanner
Spider (Crawl pages)
Active Scan (Inject payloads)
Config:
- Port: 8090
- API Key: changeme
- Max Depth: 5
- Timeout: 28800s
JSON Report (129 alerts)</b>
JSON Parser (ZAPAlertParser)
Input: ZAP JSON Report
Output: Vulnerability[]
Dedup: By alert signature
Vulnerabilities (classified)</b>
Graph Builder (NetworkX DiGraph)
Structure:
- Nodes: Vulnerabilities
- Edges: Probabilistic rules
- Type: Directed Graph
Graph (74 nodes)</b>
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)</b>
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)</b>
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)</b>
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)</b>
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 showing eight-stage pipeline from OWASP ZAP scanning to chain detection and reporting.

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

TABLE III
BASELINE VS ENHANCED SYSTEM COMPARISON

App	Baseline	Total Chains	Unique	Dedup (%)	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

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.

A critical challenge emerges from naive DFS: the algorithm 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 $\rightarrow$ Info Disclosure	1847	1
Missing Headers $\rightarrow$ XSS	2156	1
XSS $\rightarrow$ Info $\rightarrow$ 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 VI breaks down structural properties. Chains of length 3 dominate across applications, representing the most common pattern: information gathering $\rightarrow$ exploitation $\rightarrow$ impact.

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
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 three baselines: (1) standard OWASP ZAP, (2) manual analysis by experienced practitioner, (3) trivial rule-based heuristic. Table VII summarizes results.

TABLE VII
COMPARISON WITH BASELINE APPROACHES

Approach	Chains Found	Time	FP Rate
ZAP Baseline	0	8.2 min	N/A
Manual Analysis	12	4.5 hrs	0%
Simple Heuristic	156	2.1 min	67%
Our Approach	**37**	**16.6 min**	**15%**

Our approach automates chain detection (eliminating 4.5-hour manual effort), identifies 3× more chains than expert manual analysis (37 vs 12), and maintains reasonable precision (85%) compared to simple heuristic's 33%.

E. Qualitative Validation

We randomly sampled 20 unique chains for expert review by two security practitioners. Case studies confirmed exploitability:

DVWA XSS-to-Session Hijacking: Reflected XSS $\rightarrow$ Information Disclosure $\rightarrow$ Session Fixation (cumulative probability 0.68, risk score 61.2). Manual validation confirmed JavaScript injection successfully extracts session tokens enabling account takeover.

Juice Shop CORS-to-Injection: Cross-Domain Misconfiguration $\rightarrow$ Command Injection $\rightarrow$ SQL Injection (risk score 65.7). Permissive CORS policy enables external domains to trigger authenticated requests invoking vulnerable endpoints.

Among 20 chains, 17 (85%) were assessed as ”actionable and realistic.” Three false positives (15%) resulted from overly broad rules lacking runtime context validation.

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 scanners miss.

Core contributions include: (1) probabilistic rule engine encoding 53 domain-specific patterns; (2) efficient graph-based detection algorithm with intelligent pruning and deduplication; (3) normalized risk scoring combining severity, exploitability, chain length, and confidence; (4) seamless OWASP ZAP integration via REST API.

Experimental evaluation demonstrated effectiveness across 842 vulnerabilities, identifying 37 unique attack chains with 99.5-99.9% deduplication rates and processing times under 16 minutes. Detected chains represent realistic attack scenarios including XSS-enabled CSRF bypasses and configuration weaknesses facilitating SQL injection.

Future work includes: (1) machine learning to learn chain probabilities from historical attack data; (2) graph pruning algorithms for scalability beyond 1,000 vulnerabilities; (3) integration with exploitation frameworks for practical validation; (4) comparative evaluation against commercial security testing tools.

Despite limitations, our approach demonstrates feasibility and value of automated vulnerability chain detection, advancing web application security assessment and providing foundation for future research in compound threat detection.

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 *2015 IEEE 8th International Conference on 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," *International Journal of 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 Journal of 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 *2017 6th International Conference on 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 *2023 IEEE International Conference on Big Data*, pp. 1–6, 2023.

[8] S. Mauw and M. Oostdijk, "Foundations of Attack Trees," in *International Conference on 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 Transactions on 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 *Proceedings 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 Transactions on 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," *International Journal of 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 *Proceedings of the 14th International Workshop on Automation of Software Test*, pp. 55–61, 2019.