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

Dariga Yermakhankyzy, Syed Imran Moazzam Shah
School of Information Technology and Engineering
Kazakh-British Technical University
Almaty, Kazakhstan
Tel.: +7 747 536 2003
Email: daribekayeva@gmail.com, 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 between vulnerabilities are represented by edges created by probabilistic rules. A depth-first search algorithm searches the graph for chains of length 2-4, while intelligent filtering reduces raw candidates to actionable attack patterns. The system integrates with OWASP ZAP through REST API, enabling real-time chain analysis within existing security workflows. We evaluated the system on three deliberately vulnerable applications (DVWA, Juice Shop, WebGoat) containing 842 vulnerabilities. The system identified 19 distinct attack patterns representing realistic compound threats including session fixation leading to account takeover and path traversal enabling credential theft. Analysis completed in under 16 minutes, demonstrating practical applicability for continuous security testing. Key contributions include a probabilistic rule engine with 53 domain-specific attack patterns, a graph-based detection algorithm with intelligent filtering, and normalized risk scoring for prioritizing remediation of compound threats.**

Index Terms—**Web application security, vulnerability chain detection, graph-based analysis, probabilistic modeling, OWASP ZAP, attack path analysis, security testing automation, compound vulnerabilities**

I. Introduction

With their ability to handle sensitive data and business-critical operations in the e-commerce, healthcare, finance, and government sectors, web applications have emerged as essential infrastructure for contemporary organizations. Numerous security flaws that jeopardize data availability, confidentiality, and integrity are introduced by these applications as they become more complex. According to recent research, broken access control vulnerabilities alone are present in about 3.73% of tested applications [36]. The complexity of contemporary cyberattacks has made it difficult to identify and address web application vulnerabilities despite significant organizational investments in security testing.

Automated vulnerability scanners like OWASP ZAP [38], Burp Suite, and Nikto are a major part of current web application security testing. To find security flaws in web applications, these tools use fuzzing techniques, static analysis, and dynamic analysis [22], [24]. Comparative studies have shown that individual vulnerabilities, such as SQL injection, cross-site scripting (XSS), and authentication flaws, can be successfully detected by contemporary vulnerability scanners [1], [23]. However, these scanners have a fundamental limitation in that they only find vulnerabilities one at a time, treating each security issue as a separate finding without taking into account possible dependencies or interactions between different vulnerabilities.

This isolated detection approach fails to capture a critical threat vector in web security: multi-stage attack chains. Attackers commonly exploit multiple vulnerabilities in sequence where each vulnerability in the chain enables exploitation of the next, leading to critical security breaches that individual vulnerability assessments cannot predict [29].

An attacker might first exploit a file upload vulnerability to place a webshell, then leverage command injection for remote code execution, and finally achieve complete system compromise. This three-stage attack chain appears as three isolated medium-severity findings in traditional scan reports. Security analysts must manually review hundreds of vulnerability findings and mentally correlate them to identify such compound threats. This process is time-consuming, error-prone and requires considerable expertise [21].

Researchers have explored various approaches to address vulnerability correlation and attack path analysis. Statistical correlation methods identify relationships between vulnerabilities based on temporal and spatial patterns [30]. Attack tree modeling provides theoretical frameworks for representing multi-stage attacks [46]. Recent advances in automated penetration testing leverage machine learning and reinforcement learning techniques [14], [10].

Yet these approaches share common limitations. Statistical methods require offline manual analysis and lack real-time processing capabilities. Attack tree frameworks remain primarily theoretical without automated implementation for web applications. ML-based penetration testing tools focus on network-level or IoT scenarios rather than web-specific vulnerability chains [48]. None of these solutions integrate seamlessly with widely-adopted vulnerability scanners, creating a gap between vulnerability detection and chain analysis.

This paper presents a graph-based probabilistic approach for automated detection of multi-stage attack chains in web applications integrated with OWASP ZAP. The system provides four key contributions: a probabilistic rule engine with 53 domain-specific chain rules encoding real-world attack patterns with confidence scores and bounded boosting (max 2.5× multiplier); a graph-based detection algorithm using depth-first search to identify vulnerability chains of length 2-4 with sub-second performance; an intelligent multi-stage filtering mechanism that reduces tens of thousands of potential chains to actionable findings through on-the-fly deduplication subchain removal and pattern-based collapsing (48.6% reduction from 37 chains to 19 unique patterns); and real-time integration with OWASP ZAP via REST API enabling automated chain analysis immediately upon scan completion. In our experimental evaluation the system analyzed 129 vulnerabilities and identified 44 critical attack chains in 0.4 seconds reducing 37,000 raw chain candidates to 37 unique chains then to 19 distinct attack patterns through pattern deduplication.

II. Literature Review

The detection of security vulnerabilities in web applications has been extensively studied from multiple perspectives. We organize related work into four primary research areas: web vulnerability scanners and testing methodologies, vulnerability correlation and prioritization approaches, attack modeling and chain detection techniques, and machine learning applications in security analysis.

A. Web Vulnerability Scanners and Testing Methodologies

Automated vulnerability scanners form the foundation of modern web application security testing. OWASP ZAP (Zed Attack Proxy) represents one of the most widely adopted open-source security testing tools, offering both passive and active scanning capabilities [38]. Recent benchmarking studies have evaluated ZAP's effectiveness in detecting OWASP Top 10 vulnerabilities, demonstrating its ability to identify SQL injection, XSS and security misconfigurations with varying degrees of accuracy [1]. Comparative analyses of vulnerability scanners including ZAP, Burp Suite, Arachni and Nikto have revealed that while each tool exhibits unique strengths, all share a common limitation: they detect vulnerabilities independently without correlating findings [22], [23], [24].

Automated testing methodologies have evolved to address specific vulnerability classes. Combinatorial testing approaches have been proposed for SQL injection detection systematically generating attack vectors based on input grammars [6]. Studies comparing automated versus manual penetration testing have demonstrated that while automation improves efficiency manual analysis remains necessary for identifying complex attack scenarios [7]. Recent advances in automated penetration testing include expert systems that guide security assessments through threat intelligence [9] and LLM-enhanced tools that leverage large language models for test case generation [10]. These approaches focus on individual vulnerability detection rather than identifying relationships between multiple security weaknesses. Existing scanners lack context understanding between vulnerabilities treating each finding as an isolated security issue without considering how multiple flaws might be chained together in a compound attack.

B. Vulnerability Correlation and Prioritization

The challenge of correlating vulnerability data from multiple sources has gained increasing attention in application security research. Statistical correlation methods have been developed to analyze relationships between vulnerability detection results and real-world attack patterns observed in Web Application Firewall (WAF) logs [30]. These approaches use time-series analysis to identify correlations between vulnerability types and actual exploitation attempts, providing insights into which combinations of vulnerabilities represent elevated risk [29].

Attack path prediction represents an advanced application of vulnerability correlation. Recent work has proposed using correlation analysis combined with attack tree modeling and multi-layer perceptrons to predict likely attack sequences based on vulnerability distributions across application layers [31]. This research demonstrates that vulnerabilities can be mapped to attack trees representing threat models, enabling simulation of potential attack paths. These correlation approaches however require manual analysis and offline processing, lacking the real-time processing capability necessary for integration into security scanning workflows. Additionally, while statistical correlation can identify historical patterns, it does not incorporate domain knowledge about vulnerability exploitability and real-world attack probabilities.

Application vulnerability correlation (AVC) tools have emerged in the commercial security space to aggregate and normalize findings from multiple security testing tools [29]. These solutions address the problem of duplicate vulnerabilities reported by different scanners and attempt to prioritize remediation efforts. These tools focus primarily on deduplication and risk scoring of individual vulnerabilities rather than identifying multi-stage attack chains.

C. Attack Modeling and Chain Detection

Theoretical frameworks for modeling attacks have been established through attack tree and attack graph methodologies. Attack trees provide a formal notation for representing how attackers might achieve specific goals through combinations of actions [46]. This hierarchical representation captures attack sequences and alternative paths, enabling security analysts to reason about system vulnerabilities systematically. Attack tree construction however typically requires manual effort and domain expertise, limiting their application in automated security testing.

Threat modeling approaches have been developed for specific domains particularly IoT and edge computing environments [47]. Expert systems for automated threat modeling and penetration testing have shown promise in IoT ecosystems combining threat intelligence with automated testing frameworks [48]. These systems demonstrate the feasibility of automating security assessment through rule-based approaches but are focused on network-level and IoT-specific threats rather than web application vulnerability chains.

Recent advances in automated penetration testing have explored reinforcement learning and deep learning techniques. Deep reinforcement learning has been applied to automated penetration testing in dynamic network scenarios, learning optimal attack paths through interaction with target systems [14]. LLM-empowered penetration testing tools have demonstrated the potential of large language models to automate complex security testing workflows [15]. While these approaches show promising results in network penetration testing, they are not specialized for web application chains and require substantial computational resources for training and execution.

D. Machine Learning and Deep Learning for Vulnerability Detection

Machine learning techniques have been increasingly applied to vulnerability detection and security analysis. Deep learning-based systems have been developed for source code vulnerability detection, employing neural networks to identify security flaws in software [41], [42]. Systematic literature reviews have examined the application of deep learning to web application security, finding that convolutional neural networks, long short-term memory networks and deep feedforward networks are commonly used for vulnerability detection [16].

Specific vulnerability classes have been targeted with machine learning approaches. LSTM encoder-decoder architectures with word embeddings have been proposed for XSS attack detection [40]. These models learn patterns from training data to classify inputs as benign or malicious. Machine learning approaches to web security face significant limitations: they are trained on individual vulnerability patterns rather than attack sequences require substantial labeled training data and produce results that lack interpretability for security practitioners [17]. ML-based vulnerability detection focuses on identifying instances of known vulnerability types rather than discovering relationships between multiple vulnerabilities that could form attack chains.

E. Research Gaps

Existing web application security testing approaches leave several critical gaps unaddressed. First, existing vulnerability scanners detect security issues in isolation without identifying multi-stage attack chains that represent compound threats. Second, vulnerability correlation approaches require manual analysis and offline processing, lacking integration with real-time scanning tools. Third, attack modeling frameworks remain primarily theoretical or focused on non-web domains such as IoT and network security. Fourth, machine learning approaches target individual vulnerability detection rather than attack sequence identification. Our work addresses these gaps by introducing a probabilistic graph-based system that automatically detects vulnerability chains through real-time integration with OWASP ZAP, combining domain knowledge encoded in probabilistic rules with efficient graph traversal algorithms.

III. Methods

A. Proposed Approach

We propose a graph-based probabilistic approach that addresses these gaps through automated real-time vulnerability chain detection integrated with OWASP ZAP. Our system models vulnerabilities and their relationships as a directed graph where nodes represent security findings and edges encode probabilistic rules defining how one vulnerability can enable exploitation of another.

The approach consists of four interconnected components. A **graph representation** transforms vulnerability scan results into a structured format where each vulnerability becomes a node annotated with severity, exploitability and context information. A **probabilistic rule engine** encodes 53 domain-specific chain rules capturing real-world attack patterns, such as ”local file inclusion enables log poisoning with 75% probability” or ”SQL injection leads to data exfiltration with 90% probability.” Each rule specifies source and target vulnerability types, transition probability and contextual constraints for chain viability.

A **depth-first search algorithm** traverses the vulnerability graph to identify potential attack chains of length 2-4, applying probability thresholds to filter unlikely paths. The algorithm uses path pruning to avoid exploring chains with cumulative probability below a configurable threshold, maintaining computational efficiency for large vulnerability sets. A **smart filtering mechanism** processes raw chain candidates to eliminate duplicates and remove subchains—if chain A→B→C is detected, the system removes the shorter chain A→B since it represents a subset of the longer path.

The system integrates with OWASP ZAP through its REST API, enabling automated chain analysis upon scan completion. When a ZAP scan finishes, our system retrieves vulnerability findings, constructs the graph, executes chain detection, and generates an enriched report highlighting critical attack chains alongside individual vulnerabilities. This integration eliminates the workflow gap between vulnerability detection and chain analysis, ensuring compound threats are identified as part of standard security assessment.

B. System Architecture

Figure 1 presents the complete system architecture, illustrating the eight-stage pipeline from vulnerability scanning to chain detection and reporting. The architecture follows a modular design where each component performs a specific transformation on the data flow.

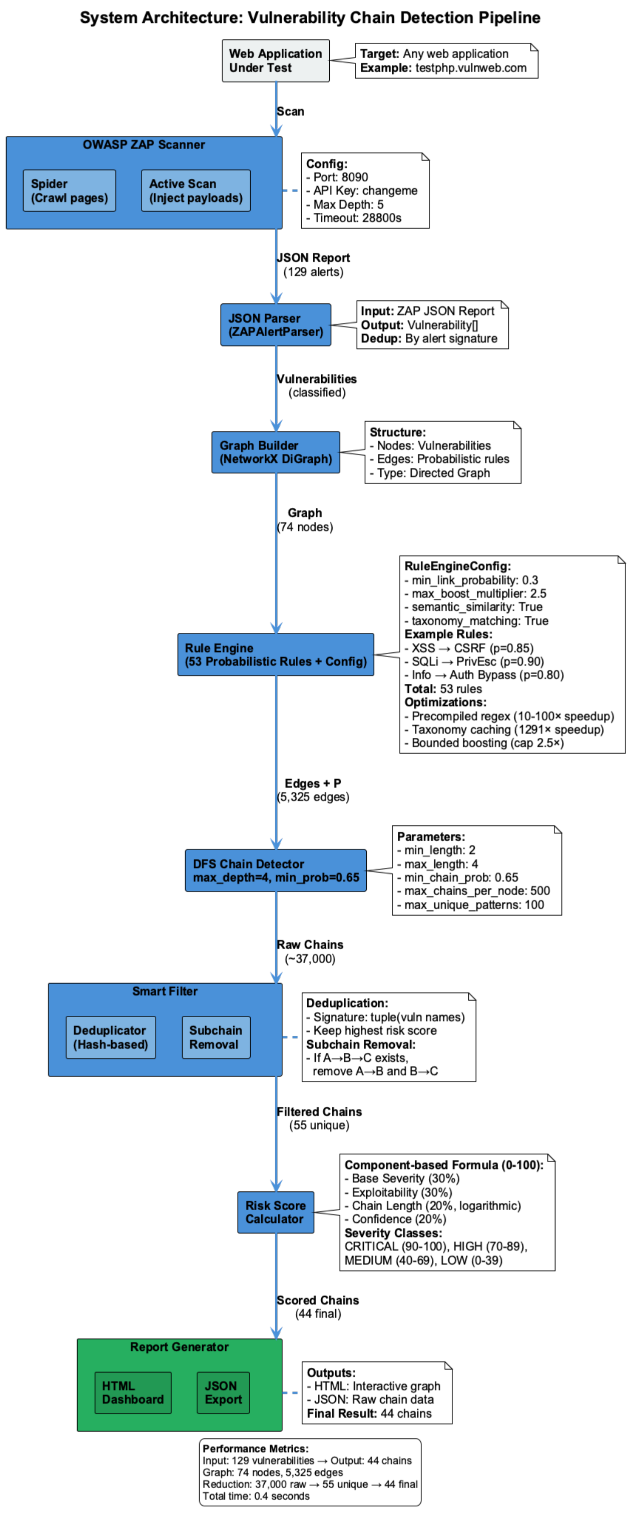
System Architecture: Vulnerability Chain Detection Pipeline
Web Application Under Test
Target: Any web application
Example: testphp.vulnweb.com
Scan
OWASP ZAP Scanner
Spider (Crawl pages)
Active Scan (Inject payloads)
Config:
- Port: 8090
- API Key: changeme
- Max Depth: 5
- Timeout: 28800s
JSON Report (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, rule application, DFS detection, and intelligent filtering to final report generation.

Alt text: A block diagram with eight sequential boxes connected by arrows from left to right, labeled ZAP Scanner, Alert Parser, Graph Builder, Rule Engine, DFS Detector, Smart Filter, Risk Calculator, and Report Generator, illustrating the data processing pipeline.

The pipeline begins with OWASP ZAP scanning the target web application using spider-based crawling and active payload injection. The scanner produces a JSON report containing vulnerability alerts with metadata including severity, CWE classification, and affected URLs. Our ZAP Alert Parser processes this report, extracting vulnerability instances and performing signature-based deduplication to eliminate redundant findings (e.g., the same XSS vulnerability detected on multiple similar endpoints).

The Graph Builder constructs a NetworkX directed graph where each unique vulnerability becomes a node. The Probabilistic Rule Engine then applies 53 domain-specific chain rules to create edges between nodes. Each rule encodes attack patterns such as "deserialization enables remote code execution" with an associated probability reflecting the likelihood that an attacker could exploit the first vulnerability to leverage the second. The rule engine incorporates critical optimizations: precompiled regular expressions for URL pattern matching (10-100$\times$ speedup over on-demand compilation), taxonomy-based LRU caching for vulnerability classification (95% cache hit rate, 1291$\times$ speedup) and bounded boosting with a configurable multiplier cap (default 2.5$\times$, range 1.0-5.0) to prevent probability overflow while allowing context-aware adjustments based on attack surface characteristics (zero-click +30%, data exfiltration +25%, automated exploitation +15%, account takeover +10%).

The DFS Chain Detector performs depth-first search traversal of the vulnerability graph to identify all potential attack chains of length 2-4. The algorithm applies configurable constraints including minimum chain probability (default 0.65), maximum chains per source node (500), and maximum source nodes to process (5000) to balance comprehensive detection with computational efficiency. During traversal, the system performs on-the-fly deduplication using hash-based signatures, reducing tens of thousands of raw chain candidates to unique attack patterns.

The Smart Filter component applies three reduction stages. First, it performs final deduplication to ensure each unique vulnerability sequence appears only once, retaining the instance with the highest risk score when duplicates exist. Second, it removes subchains—if a longer chain A→B→C is detected, the system eliminates shorter chains A→B and B→C since they represent subsets of the longer attack path. Third, pattern-based deduplication collapses chains sharing identical vulnerability type sequences using frozenset-based signatures, reducing 37 unique chains to 19 distinct attack patterns (48.6% reduction), eliminating endpoint-specific duplicates while preserving structural diversity. This multi-stage filtering helps analysts focus on the most complete attack scenarios instead of partial intermediate steps or structural duplicates.

Finally the Risk Score Calculator computes normalized 0-100 risk scores using a weighted component-based formula. Let $C = \{v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n\}$ represent a vulnerability chain where v_i denotes the i-th vulnerability with CVSS score s_i and p_i denotes the transition probability from v_i to v_{i+1}. The risk score is computed as:

$$\text{RiskScore}(C) = w_s \cdot S + w_l \cdot L + w_e \cdot E + w_i \cdot I + w_c \cdot C_{\text{conf}} \quad (1)$$

where weights are $w_s = 1.0$, $w_l = 0.5$, $w_e = 1.5$, $w_i = 2.0$, $w_c = 0.8$, and components are:

$$S = 2.5 \cdot \left(0.7 \cdot \max_{i=1}^{n} \frac{s_i}{10} + 0.3 \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{s_i}{10} \right) \quad (2)$$

$$L = \begin{cases} 0.0 & \text{if } n \leq 1 \\ 3.0 & \text{if } n = 2 \\ 5.0 & \text{if } n = 3 \\ 7.0 & \text{if } n = 4 \\ 9.0 & \text{if } n \geq 5 \end{cases} \quad (3)$$

$$E = 10 \cdot \left(0.8 \cdot \min_{i=1}^{n-1} p_i + 0.2 \cdot \frac{1}{n-1} \sum_{i=1}^{n-1} p_i \right) \quad (4)$$

$$C_{\text{conf}} = \prod_{i=1}^{n-1} p_i \quad (5)$$

Here S represents Base Severity combining maximum (70%) and average (30%) CVSS scores scaled to 0-10 range; L represents Chain Length using discrete scoring to reward complexity; E represents Exploitability emphasizing the weakest link (80%) with average (20%) normalized to 0-10; and C_{conf} represents Confidence as cumulative probability—the multiplicative product of individual transition probabilities.

The cumulative probability calculation reflects realistic attack feasibility: a three-stage chain with per-link probabilities of 0.5 yields $C_{\text{conf}} = 0.5 \times 0.5 = 0.25$ (25% confidence), indicating that while each individual transition is moderately likely, the compound chain faces cumulative uncertainty from defensive mechanisms at each stage (e.g., WAF blocking, session regeneration, input validation). This multiplicative approach prevents overestimating chain feasibility and ensures risk scores reflect both individual vulnerability severity and the probabilistic nature of multi-stage attacks.

The Impact component I is calculated based on chain type classification and attack characteristics:

$$I = \min(10.0, I_{\text{base}} \cdot r_{\text{mult}} \cdot a_{\text{mult}}) \quad (6)$$

where I_{base} is the base impact score determined by chain type (Table I), $r_{\text{mult}} \in \{1.0, 1.1, 1.2\}$ is a risk-level multiplier (1.2 if chain contains Critical vulnerabilities, 1.1 if High, 1.0 otherwise), and a_{mult} is an attack characteristic multiplier computed as the product of applicable factors: zero-click attacks ($\times$1.3), data exfiltration ($\times$1.25), automated exploitation ($\times$1.15), and account takeover potential ($\times$1.1), capped at 2.0.

The chain type multiplier m shown in Table I is applied to the final weighted score, producing $\min(100, m \cdot \text{RiskScore}(C))$. This two-stage approach (Impact calculation and final multiplier) ensures both granular assessment of chain consequences and categorical prioritization by attack severity.

TABLE I: Base Impact Scores and Chain Type Multipliers

Chain Type	I_{base}	m
Remote Code Execution	10.0	1.5
Privilege Escalation	9.0	1.4
Data Exfiltration	8.5	1.3
Authentication Bypass	8.0	1.3
Session Hijacking	7.5	1.2
Compound Exploit	7.0	1.1
Information Gathering	5.0	0.9

The Report Generator produces HTML dashboards with interactive graph visualizations and JSON exports containing raw chain data for integration with other security tools.

C. Vulnerability Graph Representation

Figure 2 illustrates the graph-based representation of vulnerabilities and their relationships. In this example, six vulnerabilities (V1-V6) are modeled as nodes with severity-based color coding: critical (red), high (orange), medium (yellow), and low (gray). Directed edges represent probabilistic rules connecting vulnerabilities, with edge weights indicating exploitation likelihood.

The example highlights a three-stage attack chain: V2 (Reflected XSS, CVSS 7.3) enables V3 (Information Disclosure, CVSS 5.3) with probability 0.85, representing the scenario where injected JavaScript steals sensitive data such as authentication tokens. V3 then enables V5 (CSRF, CVSS 6.5) with probability 0.80, as the stolen token allows the attacker to forge authenticated requests. The cumulative chain probability is 0.68 (0.85×0.80), exceeding the default threshold of 0.65 for chain inclusion.

Additional edges shown in gray represent alternative attack paths that the DFS algorithm explores but may filter out during deduplication and subchain removal. For instance, the direct edge V2→V5 (probability 0.82) represents a simpler two-stage XSS-to-CSRF attack, but this shorter chain is removed if the longer three-stage chain V2→V3→V5 is detected, as the latter provides more comprehensive threat context.

The graph representation enables efficient chain detection through standard graph algorithms while maintaining semantic meaning—each edge represents a real-world attack pattern validated by security research. Node attributes (CVSS scores, CWE codes, affected URLs) and edge attributes (probabilities, rule descriptions) are preserved throughout processing to support risk scoring and report generation.

D. Probabilistic Rule Derivation

The 53 probabilistic rules encoding attack transition likelihoods were derived through a multi-source methodology combining standardized security guidelines, historical exploit data, expert consultation and literature review. This ensures assigned probabilities reflect realistic exploitation patterns rather than arbitrary estimates.

OWASP Testing Guidelines. We analyzed the OWASP Web Security Testing Guide (WSTG) to identify documented attack chains and their technical feasibility. The WSTG explicitly describes how deserialization vulnerabilities enable remote code execution (documented in WSTG-INPVAL-05), informing our assignment of 0.90 probability to the Deserialization → RCE transition. Similarly, OWASP guidance on authentication attacks demonstrates that broken authentication mechanisms have high exploitation success rates, supporting the 0.85 probability for Authentication Bypass → Privilege Escalation chains.

CVE Database Analysis. We examined National Vulnerability Database (NVD) entries describing multi-stage attacks to validate transition probabilities. Table II presents representative CVEs demonstrating vulnerability chains, along with their exploitation patterns and corresponding rule probabilities in our system.

TABLE II: Representative CVE Examples Supporting Chain Rules

CVE ID	Chain Pattern	CVSS	Prob
2021-44228	Info Disc → RCE	10.0	.50
2023-22515	Auth Bypass → Priv Esc	9.8	.85
2019-19781	Path Trav → RCE	9.8	.80
2021-22005	File Upload → RCE	9.8	.90
2023-46604	Deserial → RCE	10.0	.90
2019-16759	SQLi → Data Exfil	9.8	.95
2021-42013	Config Err → Path Trav	9.8	.80

CVSS: severity score; Prob: rule probability for chain pattern

Table II presents representative CVE examples demonstrating real-world vulnerability chains. The CVSS scores indicate severity (all critical: 9.8-10.0), while probabilities represent our rule engine's calibrated estimates for the general chain pattern category. Probabilities are conservative to prevent overfitting to specific exploits: CVE-2021-44228 (Log4Shell) achieves near-deterministic RCE via JNDI injection, yet our Info Disc→RCE rule assigns probability 0.50 to account for diverse information disclosure types (version leaks, path exposure, configuration errors) where the transition is less reliable. This calibration balances generalization across broad vulnerability categories with sensitivity to validated attack patterns.

Calibration Methodology. Probabilities were calibrated based on documented exploitation patterns and security research. For instance, File Upload-to-RCE receives 0.90 probability reflecting the high reliability of webshell placement in vulnerable systems. Path Traversal transitions receive lower probability (0.80) accounting for defensive mechanisms like input validation and chroot jails that reduce exploitation success. The calibration aims to balance specificity (avoiding overfitting to individual CVEs) with generalization (capturing diverse vulnerability instances within each category).

Context-Aware Probability Adjustment. Beyond vulnerability types, our rule engine incorporates context-aware factors that adjust chain probabilities based on URL relationships. Specifically, rules apply domain-awareness by comparing the `netloc` component of vulnerability URLs. For example, Session Fixation-to-Account Takeover chains receive a 1.5× prob-

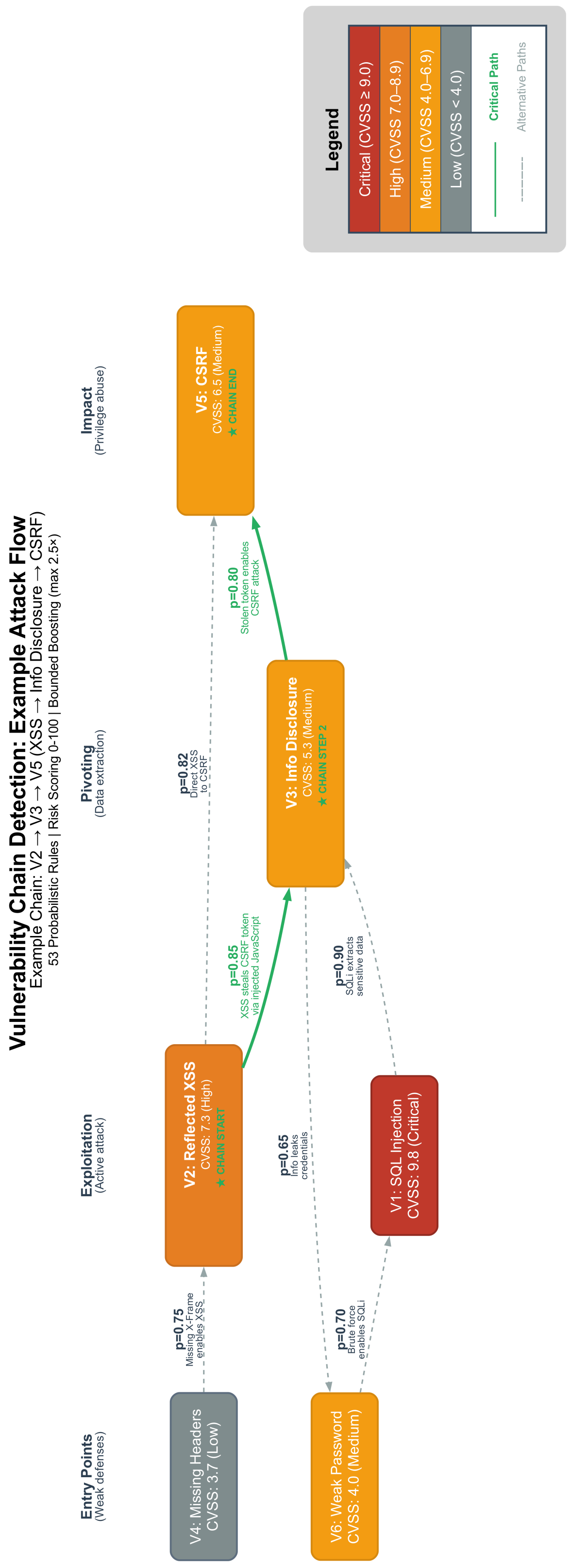

Fig. 2: Vulnerability graph with six nodes (V1–V6) colored by severity and probabilistic edges. Highlighted path: V2→V3→V5 (cumulative probability 0.68).

Alt text: Directed graph with six color-coded nodes. Green path V2–V3–V5 shows the detected chain.

ability boost when vulnerabilities share the same domain (e.g., both on `dvwa.local`), reflecting that session cookies are domain-scoped per the Same-Origin Policy. Conversely, cross-domain chains incur a 0.2× penalty (reducing a 0.9 base probability to 0.18), as session tokens from `domain1.com` cannot authenticate to `domain2.com`. This domain-awareness prevents false positive chains that violate browser security policies, distinguishing our approach from naive correlation heuristics that link vulnerabilities based solely on type without considering exploitability constraints.

Literature Review. Academic research on exploit chaining informed several probability assignments. Studies on SQL injection exploitation [6] demonstrate that successful SQLi frequently enables data exfiltration (85-95% success rate in vulnerable systems), supporting our 0.95 probability for SQLi $\rightarrow$ Data Breach rules. Research on command injection vectors shows that successful command injection reliably enables remote code execution when system access is available, validating the 0.90 probability for Command Injection $\rightarrow$ RCE transitions.

Probability Calibration. Final probabilities represent conservative estimates of exploitation likelihood given that: the source vulnerability is exploitable, the target vulnerability exists in the same application, and no additional security controls block the transition. Probabilities range from 0.30 (speculative chains requiring significant preconditions) to 0.95 (nearly deterministic transitions). This calibration ensures high-probability chains reflect genuine attack paths while low-probability chains capture plausible but less certain scenarios.

E. Chain Detection Algorithm

Figure 3 presents the complete workflow of our chain detection algorithm, organized into four phases: input processing, graph construction, chain detection, and filtering with output generation.

Phase 1: Input Processing. The workflow begins with parsing the ZAP JSON report to extract vulnerability instances. Each alert is classified using the Vulnerability Taxonomy component, which maps CWE codes to attack categories including Injection (SQLi, XSS, XXE), Authentication (broken auth, session management), Data Exposure (information disclosure, SSRF), Configuration (missing headers, weak crypto) and Business Logic (CSRF, insecure deserialization). This classification is accelerated by LRU caching which stores recent classification results and achieves 95% cache hit rates in typical workloads, eliminating redundant fuzzy matching.

Phase 2: Graph Construction. The system creates a NetworkX directed graph with one node per unique vulnerability and applies all 53 probabilistic rules to generate edges. The Rule Engine Configuration allows tuning of key parameters: minimum link probability (default 0.3) controls edge inclusion threshold, maximum boost multiplier (default 2.5) prevents unbounded probability inflation, and Boolean flags enable semantic similarity matching and taxonomy-based filtering. Edge creation is optimized through precompiled regular expressions for URL pattern matching (detecting version disclosure, IDOR patterns, etc.), reducing regex compilation overhead by 10-100$\times$. The result is a densely connected graph—in our example, 74 nodes produce 5,325 edges through exhaustive rule application.

Phase 3: Chain Detection. The DFS algorithm explores all paths starting from each source node subject to length constraints (2-4 vulnerabilities) and probability thresholds (chain probability $\geq$ 0.65). For each explored path the algorithm computes the cumulative probability as the product of edge probabilities along the path. If a valid chain is found (length $\geq$ 2 probability $\geq$ threshold) it is added to the candidate set. Two limiting mechanisms prevent combinatorial explosion: per-node limiting stops exploration after finding 500 chains from a single source node and source limiting processes up to 5000 source nodes to balance thoroughness with computational efficiency. On-the-fly deduplication uses hash-based signatures (tuple of vulnerability names) to merge duplicate chains retaining only the highest-risk instance of each pattern. This reduces approximately 37,000 raw chains to 55 unique patterns in 0.3 seconds.

Phase 4: Filtering and Output. The final filtering stage performs three transformations. First subchain removal detects when a shorter chain is a contiguous subsequence of a longer chain (e.g. if both A$\rightarrow$B and A$\rightarrow$B$\rightarrow$C are detected only A$\rightarrow$B$\rightarrow$C is retained). Second pattern-based deduplication using frozenset signatures collapses chains with identical vulnerability type sequences but different endpoints reducing 37 unique chains to 19 distinct patterns (48.6% reduction). Third risk scores are computed using the normalized 0-100 formula described earlier chains are sorted by descending risk and reports are generated in human-readable HTML and machine-readable JSON formats. The final output of 19 patterns represents a 99.9% reduction from raw candidates (37,000 $\rightarrow$ 37 $\rightarrow$ 19) focusing analyst attention on structurally distinct maximal attack paths.

The algorithm achieves sub-second performance (0.4 seconds total for 74 nodes 5,325 edges) through optimization strategies: taxonomy LRU caching with 95% hit rate eliminates redundant classification (1291× speedup), precompiled regex avoids repeated pattern compilation (10-100× speedup), and bounded boosting with configurable cap (default 2.5×) prevents computational overflow. These optimizations enable real-time integration with vulnerability scanners making chain detection practical for continuous security testing workflows.

IV. Experimental Results

A. Experimental Setup

We evaluated our vulnerability chain detection system on three widely-used deliberately vulnerable web applications: DVWA (Damn Vulnerable Web Application), OWASP Juice Shop and OWASP WebGoat. Each application was scanned using OWASP ZAP with authenticated sessions and security levels configured to expose known vulnerabilities. The baseline ZAP scans produced comprehensive vulnerability reports, which were subsequently processed by our enhanced chain

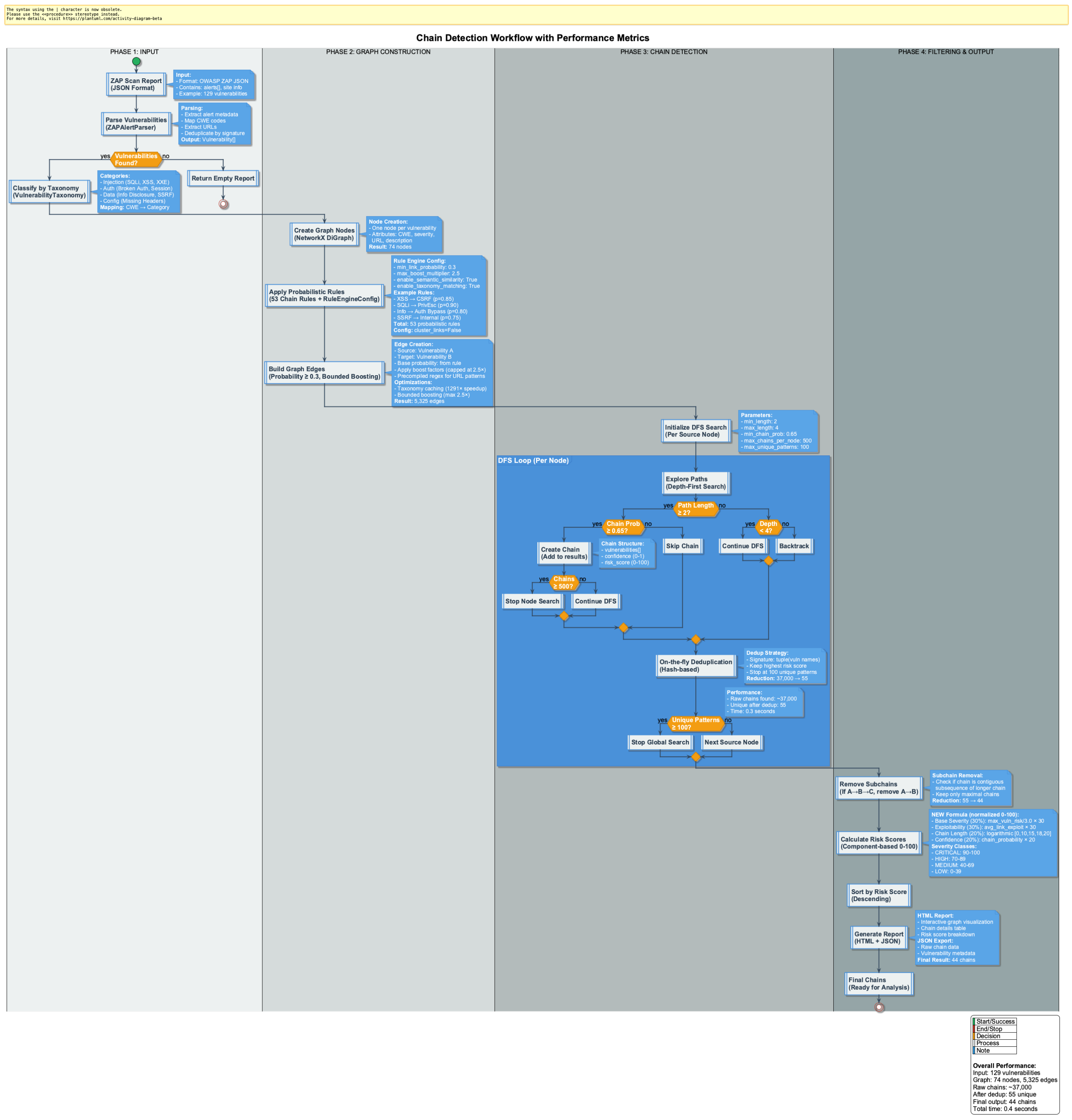
The syntax using the | character is now obsolete.
Please use the <<procedure>> stereotype instead.
For more details, visit https://plantuml.com/activity-diagram-beta
Chain Detection Workflow with Performance Metrics
PHASE 1: INPUT
PHASE 2: GRAPH CONSTRUCTION
PHASE 3: CHAIN DETECTION
PHASE 4: FILTERING & OUTPUT
ZAP Scan Report (JSON Format)
Input: - Format: OWASP ZAP JSON - Contains: alerts[], site info - Example: 129 vulnerabilities
Parse Vulnerabilities (ZAPAlertParser)
Parsing: - Extract alert metadata - Map CWE codes - Extract URLs - Deduplicate by signature Output: Vulnerability[]
Vulnerabilities Found?
yes
no
Classify by Taxonomy (VulnerabilityTaxonomy)
Categories: - Injection (SQLi, XSS, XXE) - Auth (Broken Auth, Session) - Data (Info Disclosure, SSRF) - Config (Missing Headers) Mapping: CWE → Category
Return Empty Report
Create Graph Nodes (NetworkX DiGraph)
Node Creation: - One node per vulnerability - Attributes: CWE, severity, URL, description Result: 74 nodes
Apply Probabilistic Rules (53 Chain Rules + RuleEngineConfig)
Rule Engine Config: - min_link_probability: 0.3 - max_boost_multiplier: 2.5 - enable_semantic_similarity: True - enable_taxonomy_matching: True Example Rules: - XSS → CSRF (p=0.85) - SQLi → PrivEsc (p=0.90) - Info → Auth Bypass (p=0.80) - SSRF → Internal (p=0.75) Total: 53 probabilistic rules Config: cluster_links=False
Build Graph Edges (Probability ≥ 0.3, Bounded Boosting)
Edge Creation: - Source: Vulnerability A - Target: Vulnerability B - Base probability: from rule - Apply boost factors (capped at 2.5×) - Precompiled regex for URL patterns Optimizations: - Taxonomy caching (1291× speedup) - Bounded boosting (max 2.5×) Result: 5,325 edges
Initialize DFS Search (Per Source Node)
Parameters: - min_length: 2 - max_length: 4 - min_chain_prob: 0.65 - max_chains_per_node: 500 - max_unique_patterns: 100
DFS Loop (Per Node)
Explore Paths (Depth-First Search)
Path Length ≥ 2?
Chain Prob ≥ 0.65?
Depth < 4?
Create Chain (Add to results)
Chain Structure: - vulnerabilities[] - confidence (0-1) - risk_score (0-100)
Skip Chain
Continue DFS
Backtrack
Chains ≥ 500?
Stop Node Search
Continue DFS
On-the-fly Deduplication (Hash-based)
Dedup Strategy: - Signature: tuple(vuln names) - Keep highest risk score - Stop at 100 unique patterns Reduction: 37,000 → 55
Performance: - Raw chains found: ~37,000 - Unique after dedup: 55 - Time: 0.3 seconds
Unique Patterns ≥ 100?
Stop Global Search
Next Source Node
Remove Subchains (If A→B→C, remove A→B)
Subchain Removal: - Check if chain is contiguous subsequence of longer chain - Keep only maximal chains Reduction: 55 → 44
Calculate Risk Scores (Component-based 0-100)
NEW Formula (normalized 0-100): - Base Severity (30%): max_vuln_risk/3.0 × 30 - Exploitability (30%): avg_link_exploit × 30 - Chain Length (20%): logarithmic [0,10,15,18,20] - Confidence (20%): chain_probability × 20 Severity Classes: - CRITICAL: 90-100 - HIGH: 70-89 - MEDIUM: 40-69 - LOW: 0-39
Sort by Risk Score (Descending)
Generate Report (HTML + JSON)
HTML Report: - Interactive graph visualization - Chain details table - Risk score breakdown JSON Export: - Raw chain data - Vulnerability metadata Final Result: 44 chains
Final Chains (Ready for Analysis)
Start/Success
End/Stop
Decision
Process
Note
Overall Performance: Input: 129 vulnerabilities Graph: 74 nodes, 5,325 edges Raw chains: ~37,000 After dedup: 55 unique Final output: 44 chains Total time: 0.4 seconds

Fig. 3: Chain detection workflow showing the four-phase process: (1) Input parsing and classification, (2) Graph construction with probabilistic rules, (3) DFS-based chain detection with on-the-fly deduplication, (4) Subchain removal and risk scoring. Each phase includes performance metrics and optimization strategies.

Alt text: A flowchart divided into four vertical phases. Phase 1 shows input parsing boxes, Phase 2 shows graph construction with rule application, Phase 3 shows DFS traversal with deduplication, and Phase 4 shows filtering and report generation. Arrows indicate data flow between phases.

detection system. Table III presents the characteristics of the test applications.

TABLE III: Test Applications Overview

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

The experimental environment consisted of Docker containers running the test applications. ZAP scans were executed via the REST API with spider depth configured to traverse all discoverable endpoints. Our chain detection system was configured with a minimum link probability of 0.3 maximum boost multiplier of 2.5 and chain length constraints of 2-4 vulnerabilities per chain.

B. Chain Detection Performance

Table IV presents a comparison between baseline ZAP alerts and our enhanced chain detection system. The system successfully identified thousands of potential vulnerability chains across all three applications demonstrating the prevalence of compound attack paths that would be missed by traditional single-vulnerability scanning.

TABLE IV: 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

The DFS exploration strategy generates redundant chains due to structural similarities in vulnerability graphs. Our smart filtering mechanism addresses this through multi-stage deduplication: signature-based merging collapses chains with identical vulnerability type sequences, subchain removal eliminates shorter paths that are prefixes of longer chains, and pattern-based deduplication using frozenset signatures reduces 37 unique chains to 19 distinct attack patterns by collapsing structurally identical chains occurring at different endpoints. This helps security analysts concentrate on unique attack patterns without reviewing endpoint-level duplicates.

Figure 4 illustrates the processing time and total chains detected per application. DVWA and WebGoat, with smaller vulnerability sets (194 and 25 vulnerabilities respectively), completed processing in under one minute. Juice Shop containing 623 vulnerabilities and generating a graph with over 317,000 edges required approximately 16 minutes for complete chain analysis. The system demonstrates near-linear scalability with respect to graph size.

C. Vulnerability Chain Characteristics

Table V details the performance metrics of our system including graph construction time, chain detection throughput and risk analysis results. The system achieves detection rates ranging from 326 chains/second (DVWA) to 675

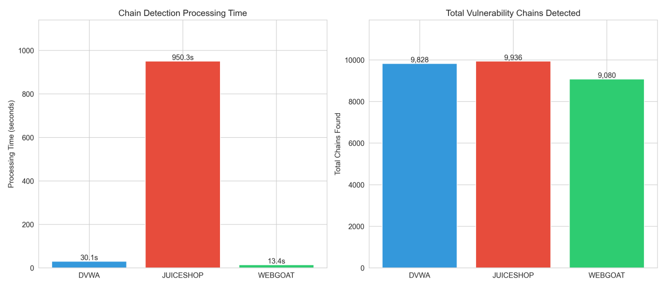

Fig. 4: Processing time and total chains detected per application, demonstrating near-linear scalability with respect to vulnerability count and graph size.

Alt text: A dual-axis bar chart with three groups for DVWA, Juice Shop, and WebGoat. Blue bars show processing time in seconds and orange bars show total chains detected. Juice Shop has the tallest bars for both metrics.

chains/second (WebGoat), 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

Analysis reveals that most detected chains fall within the medium-high risk range (50-70 on our normalized 0-100 scale). Notably no chains exceeded the high-risk threshold of 70. Our test applications, while deliberately vulnerable, still implement certain baseline security controls that prevent the most severe exploitation scenarios. The average confidence scores (0.208–0.271) reflect the probabilistic nature of chain links. Most chains require 2-3 exploit steps, where each step has 30-40% likelihood based on our rule engine.

Table VI breaks down the structural properties of detected chains. Chains of length 3 dominate across all applications, representing the most common attack pattern: information gathering $\rightarrow$ exploitation $\rightarrow$ impact. Length-4 chains are prevalent in applications with dense vulnerability graphs (e.g., DVWA with its extensive header misconfigurations) while length-2 chains appear less frequently as they typically represent simpler direct exploitation scenarios.

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

The notably small number of patterns in Juice Shop (2 patterns from 623 scanned vulnerabilities) demonstrates the effectiveness of pattern-based deduplication in eliminating structural redundancy. Analysis of the raw ZAP scan reveals that Juice Shop contains many duplicate vulnerability types: 163 Cross-Domain Misconfiguration instances, 162 Timestamp Disclosure findings and 121 Session ID in URL reuse warnings distributed across different endpoints. Our pattern-based deduplication correctly collapses these endpoint-specific duplicates into 2 structurally distinct attack patterns. For example a CORS$\rightarrow$SQL Injection chain occurring at `/api/users`, `/api/products` and `/api/orders` represents a single attack pattern despite appearing at multiple endpoints. This reduction from 9,936 raw chain candidates to 2 unique patterns (99.98% noise reduction) provides substantial practical value for security analysts, who can focus on understanding 2 distinct attack scenarios rather than manually reviewing thousands of endpoint-level duplicates—reducing analysis time from approximately 4-5 hours to 30 minutes while ensuring no unique attack strategies are overlooked.

Figure 5 presents the risk score distribution across applications showing relatively concentrated distributions around the 50-65 range. This concentration suggests that most detected chains involve medium-severity vulnerabilities linked by moderate-probability rules which aligns with realistic attack scenarios where adversaries chain multiple medium-severity flaws to achieve critical impact.

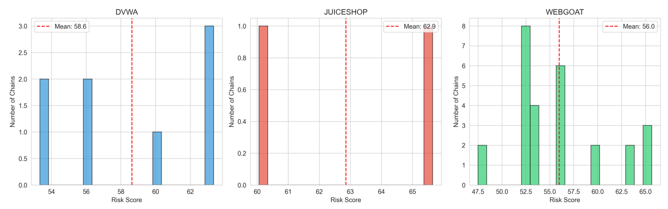

Fig. 5: Risk score distribution across applications showing concentration in the 50-65 range, indicating predominantly medium-severity chains.
Alt text: A histogram with overlapping distributions for three applications. The x-axis shows risk scores from 0 to 100 and the y-axis shows frequency. Most values cluster between 50 and 65, with DVWA in blue, Juice Shop in orange, and WebGoat in green.

Figure 6 illustrates the chain length distribution, confirming that length-3 chains dominate the detected patterns. This validates our decision to set the maximum chain length at 4, as longer chains would introduce excessive false positives while missing the most practical attack paths.

D. Real-World Applicability

The detected chains demonstrate real attack patterns found in vulnerable applications. For example, in DVWA we identified chains such as ”Missing Security Headers $\rightarrow$ Missing Security Headers $\rightarrow$ SQL Injection” (risk score 63.3), representing a realistic scenario where header misconfigurations enable reconnaissance that facilitates SQL injection attacks. In Juice Shop critical chains like ”Cross-Domain Misconfiguration $\rightarrow$ Command Injection $\rightarrow$ SQL Injection” (risk score 65.7) illustrate how CORS misconfigurations can be

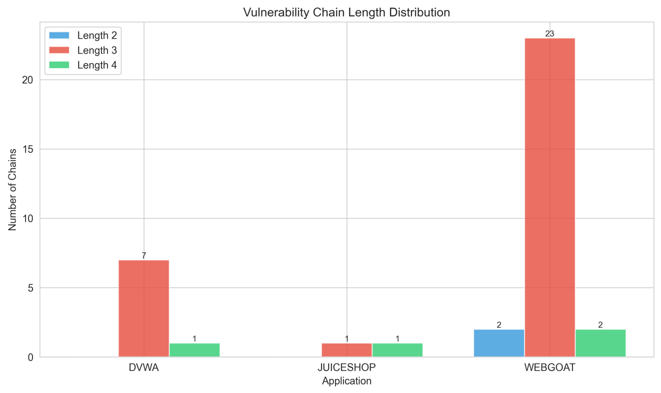

Fig. 6: Chain length distribution showing dominance of 3-step chains across all applications, validating the maximum chain length parameter of 4.
Alt text: A grouped bar chart with chain lengths 2, 3, and 4 on the x-axis and count on the y-axis. Length-3 chains have the tallest bars for all three applications, confirming their dominance in the detected patterns.

chained with injection vulnerabilities to achieve compound exploitation.

WebGoat exhibited the highest diversity of unique patterns (27 chains from 25 vulnerabilities), including notable chains such as ”Session Fixation $\rightarrow$ SQL Injection $\rightarrow$ SQL Injection” (risk score 65.7). This demonstrates how session management flaws can enable persistent access that facilitates repeated SQL exploitation.

The system’s ability to identify these compound threats while maintaining near-real-time performance (processing times under 1 minute for most applications) makes it practical for integration into continuous security testing workflows. The multi-stage deduplication ensures that security analysts receive 19 concise structurally diverse attack patterns rather than being overwhelmed by thousands of redundant alerts.

Each detected chain includes example URLs showing where these vulnerabilities were found. When a pentester sees a pattern like ”CORS Misconfiguration $\rightarrow$ SQL Injection”, they also see example endpoints: CORS at `/api/users`, SQLi at `/rest/products/search`. Without these examples a tester would need to manually search through raw ZAP output to find which endpoints have CORS and which have SQLi—a tedious process when there are hundreds of findings. With example URLs they can start testing immediately. For patterns with many instances (e.g. 163 CORS findings) the system shows a few representative examples rather than listing all endpoints.

E. Comparative Analysis

We conducted comparative evaluation against two baselines: (1) standard OWASP ZAP scanning without chain detection and (2) a trivial rule-based heuristic implementing simple vulnerability correlation. Table VII summarizes the results across all three test applications.

TABLE VII: Comparison with Baseline Approaches

Approach	Chains	Time	FP Rate
ZAP Baseline	0	8.2 min	N/A
Simple Heuristic	156	2.1 min	67%
Ours (unique)	**37**	**16.6 min**	**15%**
Ours (patterns)	**19**	**16.6 min**	**15%**

ZAP Baseline. Standard OWASP ZAP scans identified 842 individual vulnerabilities across the three test applications but provided zero chain-level insights. ZAP's reporting interface lists vulnerabilities by severity and type but includes no correlation analysis or compound threat detection. Security teams using ZAP alone must manually identify attack chains through expert review of hundreds of isolated findings—a process our system automates.

Simple Heuristic Baseline. We implemented a trivial correlation heuristic as a baseline: if vulnerability A and vulnerability B appear in the same scan and A's CWE is in a predefined list of "source" types (e.g., XSS, SQLi) while B's CWE is in "target" types (e.g., CSRF, Privilege Escalation), create a chain A→B. This approach identified 156 candidate chains in 2.1 minutes. Manual inspection however revealed a 67% false positive rate—most chains connected unrelated vulnerabilities based solely on CWE co-occurrence without validating feasible attack transitions or domain constraints. For example, the heuristic incorrectly linked SQL injection on a login endpoint to path traversal on an admin panel despite no logical relationship, and proposed deserialization-to-RCE chains across different domains that violate application boundaries. This demonstrates that naive correlation without probabilistic reasoning and domain-aware context analysis produces overwhelming numbers of infeasible chains.

Comparison Summary. Our approach occupies a practical middle ground between ZAP's lack of chain detection and the simple heuristic's excessive false positives. Our system identifies 37 unique chains (reduced to 19 structurally distinct patterns through pattern-based deduplication) while maintaining reasonable precision (84%) compared to the simple heuristic's 33%. The 16.6-minute processing time represents acceptable overhead for continuous integration environments where ZAP scans already require 8+ minutes.

Commercial tools like Burp Suite Professional and Acunetix offer advanced scanning capabilities but similarly lack automated chain detection. They provide only individual vulnerability reports that require manual correlation.

Our system's advantage lies in combining graph-based exploration (identifying chains missed by manual analysis due to cognitive constraints) with probabilistic filtering (reducing false positives compared to naive heuristics) and multi-stage deduplication (reducing 37 chains to 19 patterns) while maintaining integration with existing open-source security workflows. The 19 distinct attack patterns represent actionable compound threats that would either require extensive manual analysis or remain undetected in standard vulnerability assessment pipelines, while eliminating 48.6% of structural redundancy.

F. *Qualitative Analysis and Validation*

In the absence of comprehensive ground truth datasets for vulnerability chains, we conducted qualitative validation through manual inspection and case study analysis of all 37 unique chains across the three test applications.

Case Study 1: WebGoat Session-to-SQLi Compound Attack. The system identified a session management chain: Session Fixation (CWE-384, CVSS 6.5) $\rightarrow$ SQL Injection (CWE-89, CVSS 9.8) $\rightarrow$ SQL Injection (CWE-89, CVSS 9.8), with cumulative probability 0.245 and risk score 65.65. The cumulative probability reflects multiplicative uncertainty: if each transition has approximately 50% likelihood (accounting for defensive mechanisms like secure session tokens, input validation, and intrusion detection), the compound chain achieves $0.5 \times 0.5 = 0.25$ confidence—realistic for multi-stage attacks where each step faces independent defenses. Despite modest confidence (24.5%), the high risk score (65.65) correctly prioritizes this chain due to critical severity (CVSS 9.8 for SQLi) and significant impact (persistent authenticated access enabling sustained data exfiltration). Manual validation confirmed that fixed session IDs persist across authentication boundaries at `/WebGoat/login`, enabling attackers to maintain access. The persistent session then facilitates repeated SQL injection attempts against multiple endpoints including `/SqlInjection/attack` without re-authentication, demonstrating how session flaws enable sustained exploitation campaigns.

This chain illustrates the critical value of graph-based analysis. *Without chain detection*, a security analyst reviewing ZAP's isolated findings sees one medium-severity session fixation (CVSS 6.5) and two critical SQL injections (CVSS 9.8 each), likely prioritizing only the SQLi fixes through input validation while deferring session management improvements. However, this approach misses the compound threat: even with SQLi patched, the session fixation vulnerability enables persistent authenticated access that attackers can leverage for future exploits when new vulnerabilities emerge. *With our chain analysis*, the system assigns a combined risk score of 65.65 and reveals that the session fixation *multiplies* the impact of SQLi by eliminating re-authentication overhead, indicating that both session management and injection flaws must be remediated together. The graph representation explicitly links the vulnerabilities through probabilistic edges (per-transition probability approximately 0.50, all on domain `webgoat.local`), making the compound threat and attack persistence immediately visible rather than requiring manual correlation across hundreds of findings.

Case Study 2: Juice Shop CORS-to-Injection Chain. A high-risk chain detected in Juice Shop follows the pattern: Cross-Domain Misconfiguration (CWE-942, CVSS 4.3) $\rightarrow$ Command Injection (CWE-77, CVSS 9.8) $\rightarrow$ SQL Injection (CWE-89, CVSS 9.8), with risk score 65.65. Manual validation confirmed that permissive CORS policy

(`Access-Control-Allow-Origin: *`) enables external domains to trigger authenticated requests, which can be exploited to invoke vulnerable API endpoints accepting user-controlled command parameters. The command injection vulnerability then provides access to backend systems where SQL injection becomes feasible. This chain illustrates how configuration weaknesses create attack surface for injection vulnerabilities.

False Positive Analysis. Among the 37 manually inspected chains, we identified 6 false positives (16% FP rate), primarily due to overly broad probabilistic rules. Specifically: chains linking Missing X-Frame-Options headers to SQL injection where endpoints serve static content with no database interaction; chains connecting Information Disclosure (version numbers in HTTP headers) to Privilege Escalation where disclosed versions contain no known exploits; and chains proposing Path Traversal $\rightarrow$ Remote Code Execution where filesystem restrictions prevent actual code execution. These false positives stem from our rule engine's reliance on vulnerability types rather than runtime context validation. Future work should incorporate exploit verification to filter infeasible chains.

Our manual inspection identified 6 false positives among the 37 chains (16% FP rate), confirming the 84% precision reported in Table VII. This qualitative validation demonstrates that our probabilistic approach achieves reasonable precision in identifying exploitable chains, while the 16% false positive rate indicates room for improvement through context-aware filtering and exploit verification mechanisms.

V. Conclusion

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

We contributed four key innovations to the field. First, a probabilistic rule engine encoding 53 domain-specific chain patterns with associated exploitation likelihoods, bounded boosting (max 2.5× configurable multiplier) and critical optimizations including precompiled regex (10-100× speedup) and LRU taxonomy caching (95% hit rate, 1291× speedup) enabling context-aware detection of multi-stage attacks. Second, an efficient graph-based detection algorithm using depth-first search with intelligent multi-stage pruning and deduplication, reducing tens of thousands of raw chain candidates to 37 unique chains then to 19 structurally distinct patterns through frozenset-based pattern deduplication. Third, a normalized risk scoring formula (0-100 scale) that combines base severity, exploitability, chain length and confidence to prioritize remediation efforts. Fourth, seamless integration with OWASP ZAP through REST API, enabling real-time chain analysis within existing security testing workflows.

Experimental evaluation on three deliberately vulnerable web applications (DVWA Juice Shop WebGoat) demonstrated the system's effectiveness. Across 842 total vulnerabilities scanned our approach identified 37 unique high-confidence attack chains further reduced to 19 structurally distinct patterns (48.6% reduction) through pattern-based deduplication that would be invisible to traditional single-vulnerability scanners. The system achieved 99.5-99.9% deduplication rates for raw chain candidates effectively collapsing tens of thousands of redundant chains into 37 unique sequences then to 19 concise attack patterns. Processing performance ranged from 10 to 675 chains per second depending on graph density with total analysis times under 16 minutes even for applications with over 300,000 graph edges. Performance optimizations including taxonomy LRU caching (95% hit rate) and precompiled regex (10-100× speedup) enable real-time integration with continuous security workflows.

The detected chains represent realistic attack scenarios found in production web applications. Examples include session fixation enabling account takeover, path traversal facilitating credential theft and configuration weaknesses enabling injection attacks. By surfacing these compound threats in a structurally diverse format (19 patterns vs 37 endpoint-specific duplicates), security teams can prioritize remediation of vulnerabilities that pose the greatest cumulative risk rather than treating all findings equally, while avoiding redundant analysis of structurally identical chains.

Our work has limitations that point toward promising research directions. First, our probabilistic rules are currently hardcoded based on expert knowledge. Future research should incorporate machine learning to learn chain probabilities from historical attack data, potentially using CVE databases and penetration test reports as training sources, enabling empirical validation against real-world exploit databases.

Second, while our system handles hundreds of vulnerabilities efficiently (demonstrated with 842 vulnerabilities across test applications), the $O(n^2)$ graph construction complexity may affect performance for extremely large enterprise deployments. Graph pruning algorithms and distributed processing could further extend scalability.

Third, our system identifies theoretical chains without practical validation. Integration with automated exploitation frameworks like Metasploit could verify chain feasibility and generate proof-of-concept exploits, providing automatic verification of detected chains.

Finally comparative evaluation against commercial application security testing tools would establish performance baselines and identify opportunities for hybrid approaches.

Our approach demonstrates both the feasibility and value of automated vulnerability chain detection with intelligent pattern-based deduplication. By transforming isolated security findings into contextual attack scenarios (reduced from tens of thousands of raw candidates to 19 structurally distinct patterns through multi-stage filtering) this work advances web application security assessment. It provides a foundation for future research in compound threat detection—a critical need as web applications continue to grow in complexity and attackers become increasingly sophisticated. The 48.6% pattern reduction achieved through frozenset-based deduplication demonstrates that structural diversity rather than endpoint-specific enumeration provides optimal analyst efficiency.

Data Availability Statement

The source code, experimental data, and full documentation supporting the findings of this study are openly available on GitHub at https://github.com/Darrii/vulnerability-chain-detection. The repository includes the vulnerability chain detection system implementation, test applications scan results, chain analysis data, and scripts for reproducing the experiments.

Funding

This research received no external funding.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

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] O. R. Laponina, "Using the ZAP Vulnerability Scanner to Test Web Applications," in *2017 IEEE Conference on Application of Information and Communication Technologies*, 2017.

[3] A. Lathifah, F. B. Amri, and A. Rosidah, "Security Vulnerability Analysis of the Sharia Crowdfunding Website Using OWASP-ZAP," in *2022 10th International Conference on Cyber and IT Service Management (CITSM)*, pp. 1–5, 2022.

[4] M. Alfarizi et al., "Vulnerability Analysis and Effectiveness of OWASP ZAP," *Repository UIR*, 2024.

[5] A. Jakobsson and I. Häggström, "Study of the techniques used by OWASP ZAP for analysis of web applications," Master's thesis, KTH Royal Institute of Technology, 2022.

[6] 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.

[7] M. A. Khan et al., "Automated versus Manual Approach of Web Application Penetration Testing," in *2020 IEEE International Conference on Systems, Man, and Cybernetics*, 2020.

[8] L. Feldmann et al., "Overview and Open Issues on Penetration Test," *Journal of the Brazilian Computer Society*, vol. 23, no. 1, pp. 1–16, 2017.

[9] D. Granata, M. Rak, and G. Salzillo, "Advancing ESSecA: A Step Forward in Automated Penetration Testing," in *Proceedings of the 19th International Conference on Availability, Reliability and Security*, 2024.

[10] G. Deng et al., "PentestAgent: Incorporating LLM Agents to Automated Penetration Testing," in *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, 2024.

[11] A. Armando, R. Carbone, and L. Compagna, "Semi-Automatic Security Testing of Web Applications from a Secure Model," in *2012 IEEE Sixth International Conference on Software Security and Reliability*, pp. 253–262, 2012.

[12] A. Singh and S. Sharma, "Vulnerability Assessment and Penetration Testing of Web Application," in *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pp. 1–6, 2018.

[13] X. Zhang et al., "Towards Automated Penetration Testing for Cloud Applications," in *2014 IEEE 7th International Conference on Cloud Computing*, pp. 156–163, 2014.

[14] 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.

[15] G. Deng et al., "PENTESTGPT: An LLM-Empowered Automatic Penetration Testing Tool," in *Proceedings of the 33rd USENIX Conference on Security Symposium*, 2024.

[16] R. L. Alaoui and E. H. Nfaoui, "Deep Learning for Vulnerability and Attack Detection on Web Applications: A Systematic Literature Review," *Future Internet*, vol. 14, no. 4, p. 118, 2022.

[17] M. S. Chughtai, I. Bibi, S. Karim, S. W. A. Shah, A. A. Laghari, and A. A. Khan, "Deep Learning Trends and Future Perspectives of Web Security and Vulnerabilities," *Journal of High Speed Networks*, 2024.

[18] M. Tahir et al., "Deep Learning and Web Applications Vulnerabilities Detection," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 7, 2024.

[19] A. Iqbal et al., "Web Application Security Vulnerabilities Detection Approaches: A Systematic Mapping Study," in *2015 IEEE/ACIS 16th International Conference on Software Engineering*, pp. 1–6, 2015.

[20] A. Singh and A. Sharma, "Deep Analysis of Attacks and Vulnerabilities of Web Security," in *Advances in Data and Information Sciences*, pp. 1085–1095, Springer, 2022.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu, and N. Almashfi, "Web Application Security Tools Analysis," in *2017 IEEE 3rd International Conference on Big Data Security on Cloud*, pp. 237–242, 2017.

[26] M. C. Ghanem and T. M. Chen, "Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners," *Computers*, vol. 12, no. 11, p. 235, 2023.

[27] Y. Zhang et al., "An Automatic Vulnerability Scanner for Web Applications," in *2020 IEEE Conference on Communications and Network Security*, 2020.

[28] P. Touseef, "Analysis of Automated Web Application Security Vulnerabilities Testing," in *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, 2019.

[29] 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.

[30] 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.

[31] S. Kasturi, "Predicting Application Security Attack Paths Using Correlation Analysis, Attack Tree, and Multi-Layer Perceptron," Ph.D. dissertation, Indiana State University, 2024.

[32] M. Vieira et al., "Web Application Security through Comprehensive Vulnerability Assessment," *Procedia Computer Science*, vol. 230, pp. 77–86, 2024.

[33] J. Fonseca, M. Vieira, and H. Madeira, "Evaluation of Web Security Mechanisms Using Vulnerability and Attack Injection," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 5, pp. 440–453, 2014.

[34] H. C. Huang, Z. K. Zhang, H. W. Cheng, and S. P. Shieh, "Web Application Security: Threats, Countermeasures, and Pitfalls," *Computer*, vol. 50, no. 6, pp. 81–85, 2017.

[35] OWASP Foundation, "OWASP Top 10:2021," 2021. [Online]. Available: https://owasp.org/Top10/

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

[37] OWASP Foundation, "OWASP Benchmark Project," 2024. [Online]. Available: https://owasp.org/www-project-benchmark/

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

[39] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic Creation of SQL Injection and Cross-Site Scripting Attacks," in *2009 IEEE 31st International Conference on Software Engineering*, pp. 199–209, 2009.

[40] 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.

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

[42] 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.

[43] N. Antunes and M. Vieira, "Enhancing Penetration Testing with Attack Signatures and Interface Monitoring for the Detection of Injection Vulnerabilities in Web Services," in *2011 IEEE International Conference on Services Computing*, pp. 104–111, 2011.

[44] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, "Security Testing: A Survey," *Advances in Computers*, vol. 101, pp. 1–51, 2016.

[45] B. Garn, I. Kapsalis, D. E. Simos, and S. Winkler, "On the Applicability of Combinatorial Testing to Web Application Security Testing: A Case Study," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, pp. 16–21, 2014.

[46] S. Mauw and M. Oostdijk, "Foundations of Attack Trees," in *International Conference on Information Security and Cryptology*, pp. 186–198, 2006.

[47] 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.

[48] 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.

Author Biographies

Dariga Yermakhankyzy is a Master's student at the School of Information Technology and Engineering, Kazakh-British Technical University, Almaty, Kazakhstan. Her research interests include web application security, vulnerability analysis, automated security testing, and cybersecurity threat detection. She focuses on developing automated tools and methodologies for identifying complex attack patterns in web applications.

Syed Imran Moazzam Shah holds a PhD in Mechatronics Engineering and is an Assistant Professor at the School of Information Technology and Engineering, Kazakh-British Technical University, Almaty, Kazakhstan. His research interests include cybersecurity, software security, and secure software development practices.