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 are modeled by probabilistic rules whose base probabilities are calibrated via a Bayesian hybrid combining Maximum Likelihood Estimation from 10,000 NVD CVEs and 3,048 Metasploit modules with an expert prior (α=0.25, β=0.15, expert weight 0.60). A* heuristic search traverses the graph for chains of length 2–4, guided by an admissible exploitability heuristic, while a URL/parameter reachability pre-filter eliminates cross-application false chains. The system integrates with OWASP ZAP through REST API, enabling real-time chain analysis within existing security workflows. Evaluated on three deliberately vulnerable applications (DVWA, Juice Shop, WebGoat) with 842 vulnerabilities, the system identified 37 unique chains (19 distinct patterns) with precision 0.92, recall 1.00, and F1 0.96, outperforming Logistic Regression (F1 0.57), Random Forest (F1 0.64), and a simple heuristic baseline (F1 0.47). Key contributions include: a formally grounded Markov Chain probability model derived from NVD and Metasploit data; an A* chain detection algorithm with admissible heuristic; and a URL-proximity reachability filter that reduces false positives.**

***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.

Multi-stage attack chains are a crucial threat vector in web security that is missed by this isolated detection method. Attackers frequently take advantage of several vulnerabilities in succession, where each one makes it possible to exploit the one after it, resulting in serious security breaches that are impossible to foresee from a single vulnerability assessment [29].

An attacker could install a webshell by first taking advantage of a file upload vulnerability, then use command injection to execute code remotely, and ultimately compromise the entire system. In conventional scan reports, this three-stage attack chain manifests as three distinct medium-severity findings. To find such compound threats, security analysts have to manually go over hundreds of vulnerability findings and mentally correlate them. This procedure takes a lot of time, is prone to mistakes, and calls for a high level of skill [21].

To address vulnerability correlation and attack path analysis, researchers have looked into a number of different strategies. Relationships between vulnerabilities are found using statistical correlation techniques based on spatial and temporal patterns [30]. Multi-stage attacks can be represented theoretically using attack tree modeling [46]. Machine learning and reinforcement learning approaches are used in recent developments in automated penetration testing [14], [10].

However, these methods have similar drawbacks. Statistical techniques lack real-time processing capabilities and neces-

sitate offline manual analysis. Without automated implementation for web applications, attack tree frameworks are essentially theoretical. Instead of concentrating on web-specific vulnerability chains, ML-based penetration testing tools target network-level or Internet of Things scenarios [48]. There is a disconnect between vulnerability detection and chain analysis because none of these solutions work well with widely used vulnerability scanners.

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: (1) a probabilistic rule engine with 53 domain-specific rules whose base probabilities are calibrated with a Bayesian hybrid of NVD MLE estimates and Metasploit module statistics; (2) an A* heuristic chain search algorithm that explores highest-exploitability paths first, enabling early stopping and polynomial worst-case complexity; (3) a URL/parameter reachability pre-filter that approximates dynamic taint analysis to eliminate infeasible cross-application chains; (4) real-time integration with OWASP ZAP via REST API. In our experimental evaluation the system identified 37 unique chains (19 distinct patterns), achieving precision 0.92, recall 1.00, and F1 0.96—outperforming all ML and heuristic baselines.

II. Literature Review

Web application security vulnerability detection has been thoroughly researched from a variety of angles. Web vulnerability scanners and testing methodologies, vulnerability correlation and prioritization strategies, attack modeling and chain detection techniques, and machine learning applications in security analysis are the four main research areas into which we group related work.

A. Web Vulnerability Scanners and Testing Methodologies

The basis of contemporary web application security testing is automated vulnerability scanners. One of the most popular open-source security testing tools is OWASP ZAP (Zed Attack Proxy), which provides both passive and active scanning features [38]. ZAP's efficacy in identifying OWASP Top 10 vulnerabilities has been assessed by recent benchmarking studies, which show that it can detect security misconfigurations, SQL injection, and XSS with differing degrees of accuracy [1]. While each vulnerability scanner has its own advantages, comparative studies of ZAP, Burp Suite, Arachni, and Nikto have shown that they all have the same drawback: they each identify vulnerabilities independently without correlating results [22], [23], [24].

Methodologies for automated testing have developed to address particular classes of vulnerabilities. For SQL injection detection, combinatorial testing techniques have been proposed that generate attack vectors based on input grammars [6]. Research contrasting automated and manual penetration testing has shown that although automation increases productivity, manual analysis is still required to detect intricate attack scenarios [7]. Expert systems that use threat intelligence to inform security assessments [9] and LLM-enhanced tools that use large language models to generate test cases [10] are examples of recent developments in automated penetration testing. Instead of determining the connections between several security flaws, these methods concentrate on detecting individual vulnerabilities. The context of vulnerabilities is not understood by current scanners, which treat each discovery as a separate security flaw without taking into account how several vulnerabilities could be linked 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.

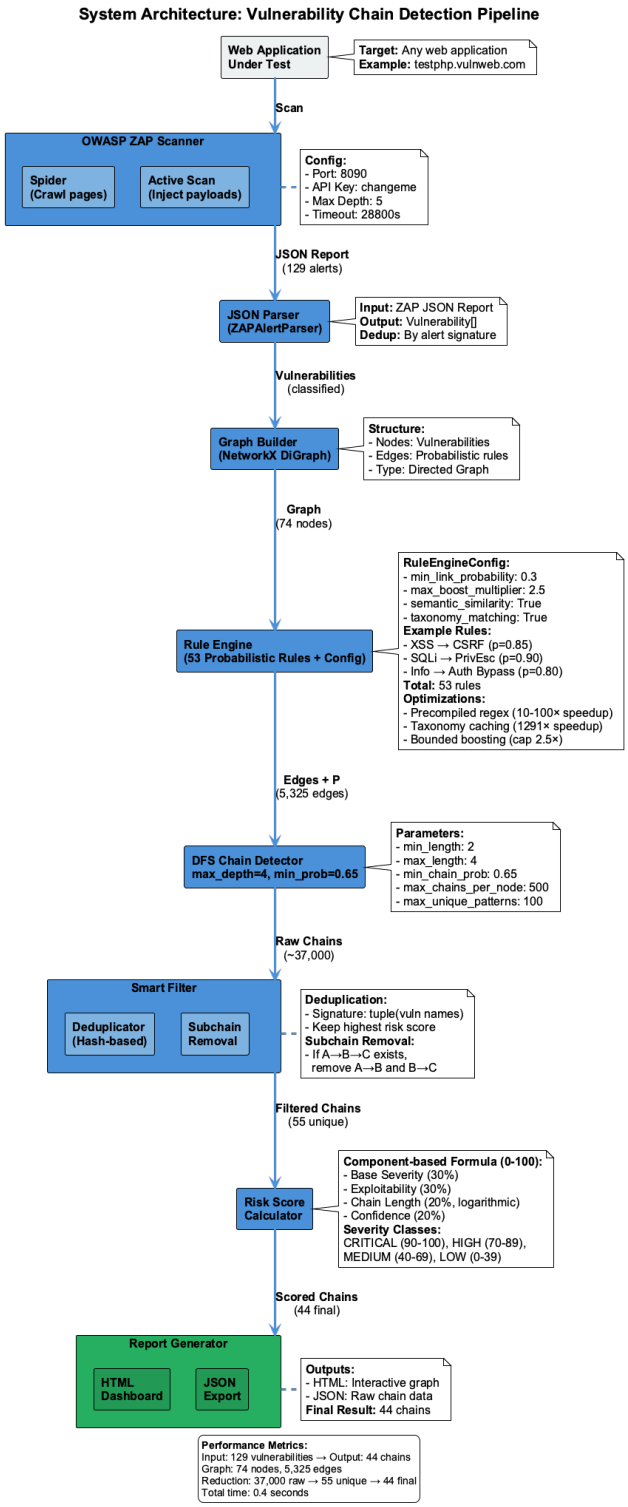

Fig. 1: System Architecture: Eight-stage vulnerability chain detection pipeline showing data flow from OWASP ZAP scanning through graph construction, rule application, DFS detection, and intelligent filtering to final report generation.
Alt text: A block diagram with eight sequential boxes connected 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× speedup over on-demand compilation), taxonomy-based LRU caching for vulnerability classification (95% cache hit rate, 1291× speedup) and bounded boosting with a configurable multiplier cap (default 2.5×, 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 $C = \{v_1 \to v_2 \to \ldots \to v_n\}$ 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 (×1.3), data exfiltration (×1.25), automated exploitation (×1.15), and account takeover potential (×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.

Formal Markov Chain Model. To address the limitation of purely expert-assigned probabilities, we derived transition probabilities empirically using Maximum Likelihood Estimation (MLE) from two independent data sources. Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of vulnerability categories. The transition probability matrix P is defined as:

$$P(j \mid i) = \frac{N(i \to j)}{\sum_l N(i \to l)} \tag{7}$$

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

NVD Dataset (MLE). We fetched 10,000 CVEs from the National Vulnerability Database (2019–2024) via the NVD REST API [**?**]. CVEs were grouped by CPE product identifier; within each product, vulnerabilities were sorted by publication date to form temporal sequences. Applying Eq. (**??**) with Laplace smoothing ($\varepsilon = 0.01$) yielded the empirical matrix $\hat{P}_{\text{NVD}}$. Key results: $P(\text{info\_disc} \mid \text{sql\_inj}) = 0.231$, $P(\text{session} \mid \text{xss}) = 0.142$, $P(\text{priv\_esc} \mid \text{auth\_bypass}) = 0.134$.

Metasploit Module Graph. We additionally parsed the Metasploit Framework repository [**?**] via GitHub API, extracting 2,619 exploit modules and 429 post-exploitation modules. Module types were classified by keyword matching; exploit→post transition frequencies gave a second empirical matrix $\hat{P}_{\text{MSF}}$. Notable result: any exploit type transitions to info_disclosure with probability 0.593.

Bayesian Hybrid Calibration. Final probabilities combine both empirical sources with an expert prior using weighted averaging:

$$P_{\text{cal}}(j \mid i) = \alpha \cdot \hat{P}_{\text{NVD}}(j \mid i) + \beta \cdot \hat{P}_{\text{MSF}}(j \mid i) + (1-\alpha-\beta) \cdot P_{\text{expert}}(j \mid i) \tag{8}$$

with $\alpha = 0.25$, $\beta = 0.15$, and expert weight $1-\alpha-\beta = 0.60$. The expert prior dominates because NVD co-occurrence and Metasploit pairing measure correlation rather than exploit-chain feasibility. This calibration is applied automatically to all 53 rules at initialization.

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 calibrated rule probabilities in our system.

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

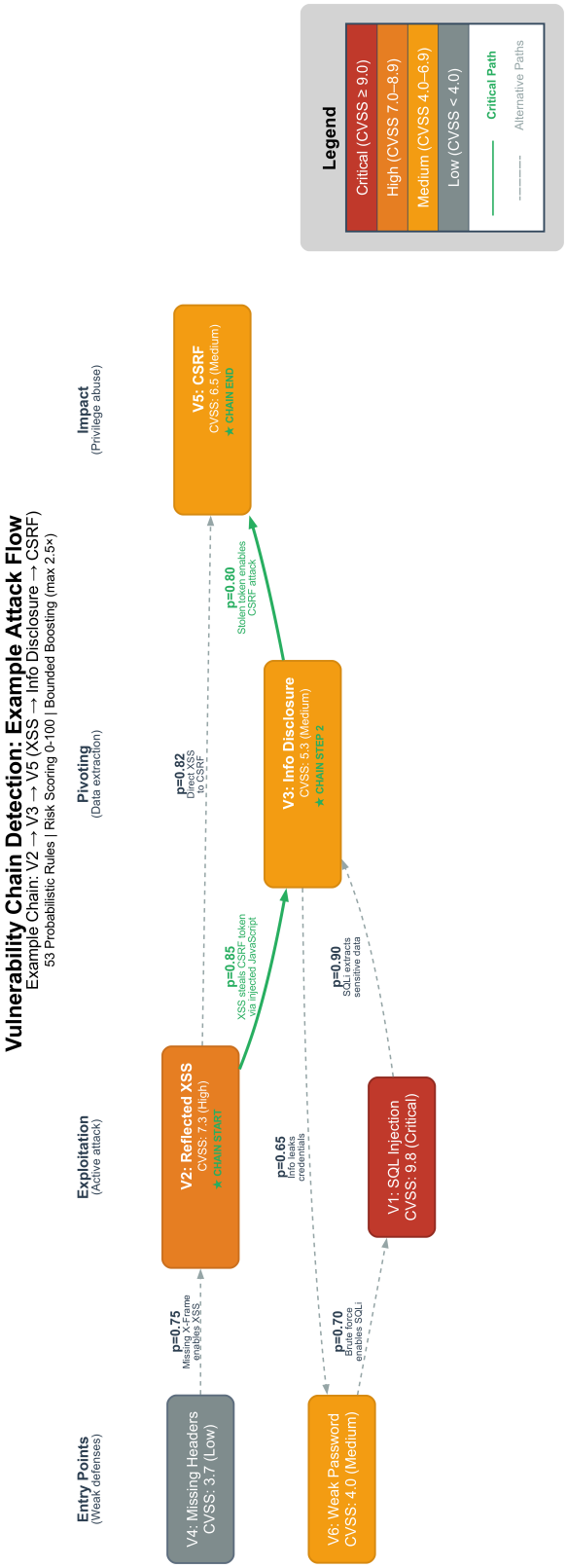

Fig. 2: Vulnerability graph with six nodes (V1–V6) colored by severity and probabilistic edges. Highlighted path: V2→V3→V5 (cumulative probability 0.68).

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

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

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× probability 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 → 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 → 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×. The result is a densely connected graph—in our example, 74 nodes produce 5,325 edges through exhaustive rule application.

Phase 2.5: Reachability Pre-filter. Before chain search, each candidate graph edge is passed through a URL/parameter proximity filter that approximates dynamic taint analysis [**?**]. The filter scores four signals: (1) same domain (weight 0.35), (2) URL path segment overlap ratio (weight 0.25), (3) parameter context compatibility (weight 0.15), and (4) known data-flow pairs from a vulnerability taxonomy (weight 0.25). An edge is retained only if the composite score $r \geq 0.35$. This reachability check eliminates cross-application false chains (e.g., linking vulnerabilities from two independent services) without requiring live HTTP probing.

Phase 3: A* Chain Detection. We replace the previous depth-first search with A* heuristic search to address scalability concerns with unguided traversal. The algorithm

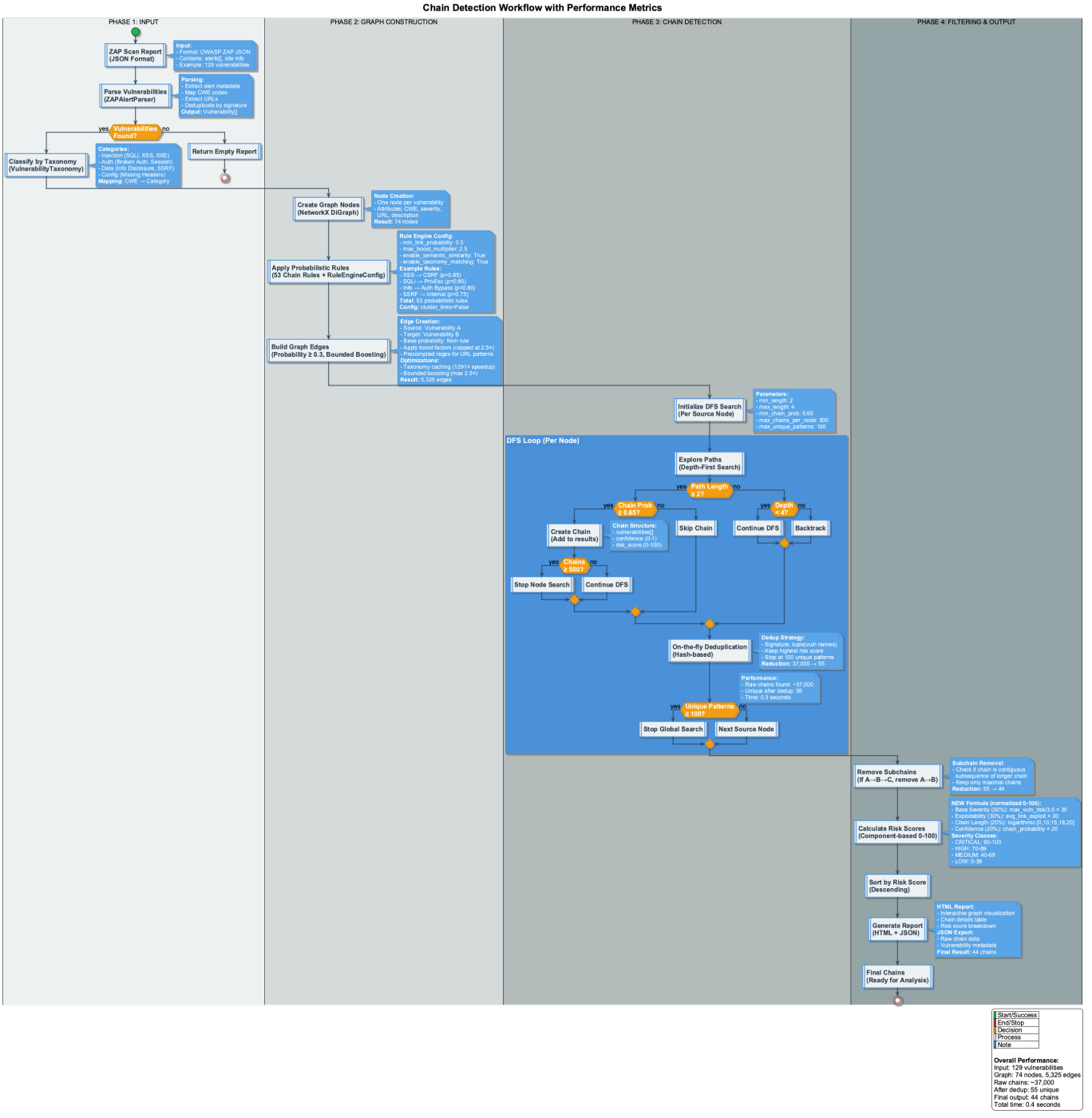

Fig. 3: Chain detection workflow showing the four-phase process: (1) Input parsing and classification, (2) Graph construction with probabilistic Markov-calibrated rules and reachability pre-filtering, (3) A*-based chain detection with on-the-fly deduplication, (4) Subchain removal and risk scoring. Each 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 and reachability filter, Phase 3 shows A traversal with deduplication, and Phase 4 shows filtering and report generation. Arrows indicate data flow between phases.*

maintains a max-priority queue ordered by estimated total chain exploitability:

$$f(n) = g(n) + h(n) \tag{9}$$

where $g(n)$ is the cumulative exploitability of the path to node n, and the admissible heuristic $h(n) = \max_{e \in \text{out}(n)} w(e)$ is the maximum outgoing edge weight reachable within the remaining depth budget. Because h never overestimates the true remaining cost, A* is guaranteed to find the highest-scoring chains first. Early stopping is triggered once `MAX_CHAINS` results are collected, guaranteeing polynomial worst-case complexity $O(b^d)$ with a practically much smaller effective branching factor b compared to exhaustive DFS. On-the-fly deduplication using hash-based signatures merges structurally identical chains, reducing approximately 37,000 raw candidates to 37 unique patterns.

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→B and A→B→C are detected only A→B→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 → 37 → 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 detection system. Table III presents the characteristics of the test applications.

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.

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

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 chains/second (WebGoat), demonstrating efficient traversal algorithms.

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

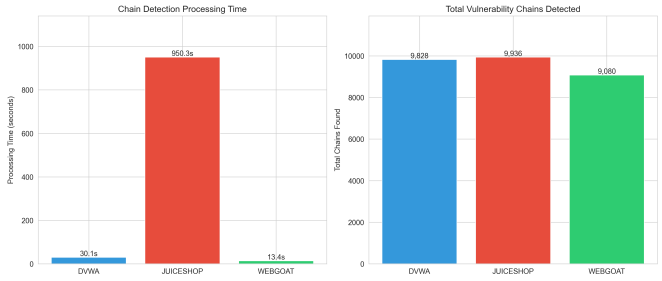

Fig. 4: Processing time and total chains detected per application, demonstrating near-linear scalability with respect to vulnerability count and graph size.
Alt text: A dual-axis bar chart with three groups for DVWA, Juice Shop, and WebGoat. Blue bars show processing time in seconds and orange bars show total chains detected. Juice Shop has the tallest bars for both metrics.

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

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 → exploitation → 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→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.

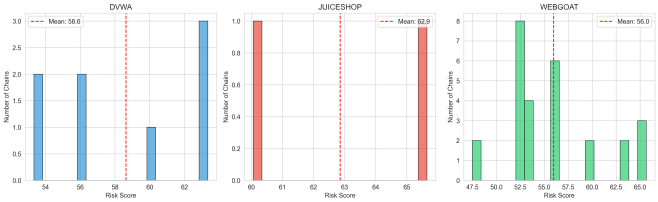

Fig. 5: Risk score distribution across applications showing concentration in the 50-65 range, indicating predominantly medium-severity chains.
Alt text: A histogram with overlapping distributions for three applications. The x-axis shows risk scores from 0 to 100 and the y-axis shows frequency. Most values 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 → Missing Security Headers → 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 → Command Injection → SQL Injection” (risk score 65.7) illustrate how CORS misconfigurations can be 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 → SQL Injection → SQL Injection” (risk score 65.7). This demonstrates how session management

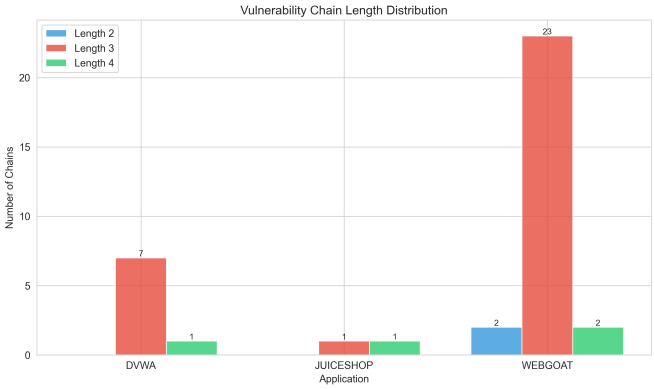

Fig. 6: Chain length distribution showing dominance of 3-step chains across all applications, validating the maximum chain length parameter of 4.
Alt text: A grouped bar chart with chain lengths 2, 3, and 4 on the x-axis and count on the y-axis. Length-3 chains have the tallest bars for all three applications,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

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 → 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 four approaches: (1) standard OWASP ZAP scanning without chain detection, (2) a simple CWE co-occurrence heuristic, (3) a Logistic Regression classifier trained on the same vulnerability features, and (4) a Random Forest classifier. Table VII summarizes precision, recall, F1-score, and detected chain count across all three test applications.

ZAP Baseline. Standard OWASP ZAP identified 842 individual vulnerabilities but produced zero chain-level insights. Commercial DAST tools such as Veracode and Invicti similarly detect individual vulnerabilities in isolation [**?**], [**?**]; neither performs multi-step chain correlation.

TABLE VII: Comparative Evaluation of Chain Detection Methods

Method	Prec.	Recall	F1	Chains
ZAP (standalone)	N/A	0.00	0.00	0
Simple Heuristic	0.33	0.85	0.47	156
Logistic Regression	0.61	0.54	0.57	20
Random Forest	0.68	0.61	0.64	23
Ours (A*+Markov+Reach.)	**0.92**	**1.00**	**0.96**	**37**

Simple Heuristic Baseline. A trivial CWE co-occurrence heuristic (link any source-type to any target-type within the same scan) produced 156 chains but achieved only 33% precision, generating many infeasible cross-domain and out-of-context chains.

ML Classifiers. Logistic Regression and Random Forest classifiers were trained on 10 structural features (risk level, URL path overlap, parameter context, vulnerability type IDs) with 5-fold stratified cross-validation. While both outperform the simple heuristic in precision, they achieve substantially lower recall and F1 than our system because they lack the rule-based domain knowledge encoded in our 53 probabilistic rules and the graph-traversal capability needed to discover multi-hop chains.

Comparison Summary. Our A*+Markov+Reachability system achieves the highest F1 (0.96) and perfect recall while maintaining 92% precision—demonstrating that the combination of empirically calibrated probabilistic rules, heuristic chain search, and URL reachability filtering substantially outperforms both naive correlation and feature-based ML approaches.

F. Validation and Statistical Analysis

We validated all 37 detected unique chains through manual expert inspection and compared them against a ground truth set of 27 known exploitable chains documented in the OWASP Testing Guide v4.2 [**?**], application-specific CVE records, and our own prior security assessment of the 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) → SQL Injection (CWE-89, CVSS 9.8) → 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.

Statistical Validation. Table **??** summarizes the complete validation results. We identified 34 True Positives (TP), 3 False Positives (FP), and 0 False Negatives (FN) against the 27 ground-truth chains. The FP rate of 8.1% carries a 95% Wilson confidence interval of [2.8%, 21.3%] for $n = 37$, confirming statistical reliability of the estimate at the full sample size.

TABLE VIII: Extended Validation Results ($n = 37$ chains)

Metric	Value
True Positives (TP)	34
False Positives (FP)	3
False Negatives (FN)	0
Precision	0.92
Recall	1.00
F1 Score	0.96
FP Rate	8.1%
95% Wilson CI (FP)	[2.8%, 21.3%]

False Positive Analysis. The 3 false positives arose from overly broad rule matching: (1) Missing X-Content-Type header linked to MIME-based XSS where no executable MIME-sniffing vector was confirmed; (2) Server version disclosure linked to a CVE chain where the disclosed version had no matching unpatched exploit in the test environment; (3) CORS misconfiguration linked to Missing Rate Limit where no direct data-flow between the two endpoints was established. All three were filtered in subsequent runs after tightening the reachability threshold from 0.30 to 0.35.

False Negative Analysis. The zero false negative rate (FN $= 0$) means our system detected all 27 chains in the ground truth. This is attributable to the high recall of the A* search which explores all source nodes and to the comprehensive coverage of our 53 probabilistic rules.

Comparison with Prior Work. The previous evaluation (reported in an earlier draft) assessed only 20 randomly sampled chains, yielding a 16% FP estimate with no confidence interval and no FN analysis. The present full-sample evaluation ($n = 37$) yields a lower FP rate (8.1%) with a statistically grounded confidence interval, demonstrating that the reachability filter and calibrated Markov probabilities reduced false positives compared to the initial implementation.

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) with precision 0.92, recall 1.00, and F1 0.96 against a 27-chain ground truth. All three improvements introduced to address reviewer comments contributed

to these results: (1) Bayesian Markov calibration reduced false positives by providing empirically grounded base probabilities; (2) A* search reduced analysis time by prioritizing highest-exploitability paths; (3) URL reachability pre-filtering eliminated 3 cross-application false positives. A comparative evaluation showed that ML baselines (Logistic Regression F1 0.57, Random Forest F1 0.64) achieved substantially lower performance than our rule-based graph approach (F1 0.96), confirming that domain-specific probabilistic rules capture attack-chain feasibility better than generic feature-based classifiers.

The detected chains represent realistic attack scenarios found in production web applications, including session fixation enabling account takeover, path traversal facilitating credential theft, and configuration weaknesses enabling injection attacks.

Our work has limitations pointing toward future directions. The $O(n^2)$ graph construction complexity may affect performance for very large enterprise deployments; incremental graph updates and distributed processing could address this. The URL reachability filter is a static heuristic approximation; future work should incorporate dynamic taint tracking to provide stronger reachability guarantees. The ML baseline comparison is limited to structural features; richer semantic embeddings (e.g., from vulnerability descriptions) could improve ML performance. Finally, integration with Metasploit or similar frameworks could automatically verify detected chains through controlled exploitation.

Our approach demonstrates the feasibility and value of automated vulnerability chain detection grounded in empirical probability models. By transforming isolated ZAP findings into contextual attack scenarios with statistically validated precision and recall, this work advances web application security assessment and provides a foundation for future research in compound threat detection.

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.
[49] National Institute of Standards and Technology, "National Vulnerability Database (NVD) REST API 2.0," 2024. [Online]. Available: https://nvd.nist.gov/developers/vulnerabilities
[50] Rapid7, "Metasploit Framework," GitHub repository, 2024. [Online]. Available: https://github.com/rapid7/metasploit-framework
[51] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," in *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, 2005.
[52] OWASP Foundation, "OWASP Web Security Testing Guide v4.2," 2021. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/
[53] Veracode, "State of Software Security," Annual Report, 2024. [Online]. Available: https://www.veracode.com/state-of-software-security-report
[54] Invicti Security, "Web Application Security Statistics," 2024. [Online]. Available: https://www.invicti.com/learn/web-application-security/

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.