© AUG 2026 | IRE Journals | Volume 10 Issue 2 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I2-1722336

Evolution of Web Application Attacks: A Systematic Analysis of the Current Threat Landscape and Emerging Security Challenges

JOSE MARCELITO D. BRIGOLI[1], TEODORO B. COMAYAS JR.[2], IRENE I. EDA[3], BERNIE C. EMPACES[4], JENEL T. TIAD[5], SERAFIN C. PALMARES[6], KRISTINE T. SOBERANO[7]
[1]*Ethical Hacking and Forensics, Master in Information Technology*
[2]*State University of Northern Negros, Sagay City, Negros Occidental*

Abstract *- The rapid explosion of web application capabilities over the past ten years has fundamentally redesigned how applications are delivered, simultaneously introducing an intricate, multifaceted attack surface that continues to evolve. Standard vectors, long thought to be understood—such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF)—still exist as potent hazards, yet they are increasingly overshadowed. Emerging cloud-native architectures, serverless delivery mechanisms, microservices, and AI-driven automation introduce entirely new categories of subtle, deeply embedded vulnerabilities. This research evaluates how these threats have metastasized and traces the origins of modern security vectors to determine if established defensive protocols remain effective against increasingly complex modern exploitation tactics. We conducted a strict Systematic Literature Review (SLR) structured by PRISMA 2020 guidelines, filtering an extensive initial pool of 885 records down to 62 core sources published between 2015 and 2026. This foundational dataset synthesizes observations from 44 peer-reviewed empirical studies alongside analysis from 18 key cybersecurity frameworks and official threat intelligence reports, incorporating guidance from standards bodies including OWASP, NIST, and ISO/IEC. The synthesized evidence reveals a definitive and strategic maturation in adversarial approach: threat actors are abandoning isolated, single-vulnerability exploits. They are instead executing prolonged, multi-stage campaigns that specifically leverage the trust relationships found in interconnected software ecosystems. While SQLi, XSS, and authentication weaknesses remain critical and frequent (identified within our 12 primary attack categories), a steep rise in complex, multi-stage exploit chains, AI-assisted reconnaissance, API breaches, and software supply chain compromises represents the new operational normal for adversaries. Furthermore, our analysis indicates that traditional defensive frameworks like secure development lifecycles, Zero Trust Architecture, DevSecOps, and Web Application Firewalls (WAFs) are no longer sufficient in isolation. Their mitigation capacity works only when supported by continuous, real-time context-aware monitoring and truly dynamic risk management, establishing an empirical baseline for architecting resilient security posturing that can keep pace with accelerating innovation.*

Index Terms *- Cybersecurity, Emerging Threats, Systematic Literature Review, Web Application Attacks, Web Application Security*

I. INTRODUCTION

Web software looks almost nothing like it did ten years ago. Instead of serving basic HTML pages, current systems handle critical operations—banking transactions, hospital records, e-commerce checkouts, and cloud backend tasks. But as organizations migrated core operations to the web, they created an unprecedented target. Modern threat actors rarely rely on simple, single-payload attacks. Instead, they hit complex application logic, targeting exposed API endpoints, serverless tasks, and microservice setups.

Even with thousands of published papers on web security, the broader academic picture remains oddly fragmented. A lot of studies dig deep into isolated bugs—testing how a Web Application Firewall stops an SQL injection or a Cross-Site Scripting script in a sandbox. What gets left out is how these long-standing flaws merge with newer attack vectors across modern cloud software pipelines. That leaves security engineers with plenty of data on specific bugs, but very little guidance on how threat actors string these vulnerabilities together in full real-world campaigns.

Even with thousands of published papers on web security, the broader academic picture remains oddly fragmented. A lot of studies dig deep into isolated bugs—testing how a Web Application Firewall stops an SQL injection or a Cross-Site Scripting script in a sandbox. What gets left out is how these long-standing flaws merge with newer attack vectors across modern cloud software pipelines. That leaves security engineers with plenty of data on specific bugs, but very little guidance on how threat actors string these vulnerabilities together in full real-world campaigns.

We put together this Systematic Literature Review (SLR) to bridge that gap. Following the PRISMA 2020 standard, we collected data across peer-reviewed studies, industry threat intelligence, and core security frameworks (OWASP, NIST, and ISO/IEC) published from 2015 through 2026. The goal was straightforward: map out how web attack patterns shifted over time, figure out why modern exploits succeed despite active defenses, test how existing defensive controls handle actual threats, and identify where current research still falls short.

Laying out this empirical timeline gives developers, system administrators, and security teams a realistic benchmark for risk evaluation. Rather than relying on static threat models, this paper provides concrete data to help teams build defensive strategies that reflect actual adversary behavior.

II. REVIEW OF RELATED LITERATURE

2.1 Introduction

Traditional network perimeters no longer protect modern applications. When enterprise engineering teams shifted away from physical, single-server setups to build on public cloud environments, distributed microservices, public API endpoints, and continuous integration/continuous deployment (CI/CD) pipelines, the overall exposure surface multiplied overnight. Security scholarship evolved in response. Rather than focusing solely on localized code defects—such as SQL Injection (SQLi), Cross-Site Scripting (XSS), or Cross-Site Request Forgery (CSRF)—recent literature emphasizes systemic architectural risks. These include third-party software supply chains, identity and access management, secure software development lifecycles (SSDLC), Zero Trust frameworks, and real-time observability across production environments (Open Worldwide Application Security Project [OWASP], 2021, 2023; National Institute of Standards and Technology [NIST], 2024; Verizon, 2025).

A notable limitation in current application security research is its heavy thematic isolation. Empirical papers routinely evaluate a single exploit method or test one defensive tool inside a sterile lab environment. Far fewer studies examine how legacy code defects persist inside cloud-native stacks or trace how threat actors chain multiple vulnerabilities across complex, multi-tenant software ecosystems. This disconnect leaves security engineers without a practical framework for long-term threat modeling. Our review resolves this gap by aggregating findings across academic literature, threat intelligence feeds, international standards, and breach statistics, framing web application defense as an integrated, multi-layered system.

2.2 Evolution of Web Applications

Redesigning software delivery models over the past twenty years reshaped application security priorities. Initial web applications functioned primarily as static documents, rendering basic HTML with negligible server-side processing or user input capabilities. Consequently, security concerns during this early era remained narrow, centered mostly on web server configuration errors, unauthenticated file directory traversal, and loose file system permissions.

The transition to Web 2.0 replaced static pages with dynamic, stateful platforms driven by client-side JavaScript, AJAX, server-side scripting (PHP, ASP.NET), and backend relational databases. While these frameworks unlocked interactive user experiences, they introduced massive security liabilities. Dynamic business logic and unchecked user inputs enabled widespread application flaws. Injections, script execution attacks, and session forgery became endemic during this era due to missing input validation rules, improper output escaping, and weak session cookie management.

Modern web applications bear almost no structural resemblance to early dynamic sites. Present-day systems function as distributed digital ecosystems. Engineering teams build applications using cloud

providers, microservice clusters, serverless workloads, third-party APIs, machine learning engines, and extensive open-source package trees. Although these architectures accelerate release velocity, they introduce complex failure chains. An exposed API token, an over-privileged service account, or a compromised open-source package can trigger platform-wide security incidents. Classic application flaws persist, but modern threat actors increasingly target identity infrastructure, cloud configurations, and software supply chains.

2.3 Traditional Web Application Attacks

Classic application-layer flaws still account for a major share of corporate data breaches globally (Verizon, 2025). Decades after initial documentation, database command injection, client-side script execution, and request forgery remain primary access points for threat actors. Their continued presence reflects operational friction: technical debt in legacy codebases, compressed release schedules, skipped code reviews, and inconsistent adherence to secure coding standards.

2.3.1 SQL Injection (SQLi)

Database command injection poses a persistent threat to data confidentiality and storage integrity. This flaw occurs when unsanitized user input merges directly into raw database commands, altering intended query logic. Empirical studies document severe outcomes from successful exploits, including unauthorized data exfiltration, record corruption, administrative account takeover, and host execution. Modern development frameworks mitigate injection through parameterized queries, prepared statements, and Object-Relational Mapping (ORM) abstractions. However, these protections fail when applied inconsistently across dev teams. Audits routinely catch injection defects in custom legacy modules and enterprise tools written without strict input validation rules.

2.3.2 Cross-Site Scripting (XSS)

Client-side script injection remains a prevalent threat that allows attackers to run malicious code within a target user's browser session. OWASP categorizes these attacks into Stored, Reflected, and DOM-based variants depending on payload delivery and execution paths (OWASP, 2021). Depending on application context, execution leads to session hijacking, credential harvesting, browser compromise, or site defacement. Modern client-side frameworks (React, Angular, Vue.js) provide default defenses through automatic contextual output encoding. However, vulnerabilities re-emerge whenever applications bypass framework rendering rules, manipulate raw DOM elements unsafely, load untrusted third-party packages, or omit strict Content Security Policies (CSP).

2.3.3 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery targets authenticated state by tricking a victim's browser into issuing unauthorized requests against a trusted web application. Early studies evaluated traditional, cookie-based session architectures such as retail or online banking portals. However, widespread adoption of Single Sign-On (SSO), OAuth workflows, and stateless token API authentication created modern CSRF variations (OWASP, 2023). In these environments, misconfigured token enforcement or loose Cross-Origin Resource Sharing (CORS) policies expose endpoints to unauthorized action execution. Standard mitigations—including anti-forgery tokens, SameSite cookie flags, multi-factor authentication, and strict origin validation (OWASP, 2021)—neutralize CSRF, but configuration mistakes routinely re-expose application endpoints.

2.3.4 Synthesis of Traditional Web Application Attacks

Traditional flaws persist because recognized defensive controls are implemented unevenly across software pipelines (Verizon, 2025). Although operating at different technical layers—databases, browser contexts, and authenticated sessions—they stem from shared engineering gaps: absent input validation, unsafe default choices, and missing security checks. In modern cloud setups, threat actors frequently leverage these classic flaws as initial footholds to pivot into APIs, microservices, and internal enterprise networks.

2.4 The OWASP Top 10 as the Foundation of Web Application Security

The Open Worldwide Application Security Project (OWASP) Top 10 serves as the primary industry reference for prioritizing application-layer security risks. Derived from global threat datasets and expert consensus, it provides developers, auditors, and security analysts with a standardized baseline for

vulnerability assessment and training curriculum design.

Reviewing changes across OWASP Top 10 releases reveals major shifts in software architecture and adversary methods. Early editions focused heavily on isolated, code-level bugs like raw injection, basic scripting flaws, and surface-level misconfigurations. Consequently, early defensive research emphasized input sanitization and perimeter patch management.
The OWASP Top 10 (2021) edition shifted toward systemic architecture risks, elevating categories like Broken Access Control, Cryptographic Failures, Identification/Authentication Failures, Software/Data Integrity Failures, Security Logging Failures, and Server-Side Request Forgery (OWASP, 2021). This reorientation reflects the growing reliance on identity systems, open-source dependencies, and cloud management layers. Recent papers stress that while OWASP provides a fundamental baseline, comprehensive defense requires combining it with the OWASP API Security Top 10, MITRE ATT&CK, and the NIST Cybersecurity Framework.

2.5 Emerging Web Application Threat Landscape
Contemporary cybersecurity literature frames API abuse, cloud misconfigurations, software supply chain compromises, and AI-driven threats as connected elements of an expanding attack surface. The underlying vulnerability across all four vectors is structural interdependency: modern applications rely heavily on external code, shared infrastructure, and third-party identity providers that carry risk across enterprise boundaries.

2.5.1 API Security
APIs link modern web, mobile, and microservice architectures, but they also expose backend business logic directly to internet traffic. Common risk patterns include Broken Object Level Authorization (BOLA), broken authentication checks, rate-limiting failures, and unmonitored "shadow" APIs (OWASP, 2023). Because signature-based Web Application Firewalls (WAFs) struggle to identify context-aware logic abuse, recent studies advocate for automated API endpoint discovery, schema validation, fine-grained access controls, and continuous behavioral traffic monitoring.

2.5.2 Cloud Misconfiguration
Cloud migrations shift a major portion of the attack surface away from custom source code toward identity access management (IAM) and infrastructure settings. Unsecured cloud storage buckets, over-privileged service accounts, permissive security groups, and public management interfaces often expose sensitive assets without requiring complex exploit payloads (Verizon, 2025). Consequently, contemporary literature treats least-privilege IAM policies, automated infrastructure-as-code scanning, and continuous compliance checks as core application security practices.

2.5.3 Software Supply Chain Attacks
Software supply chain compromises represent a severe threat to modern software engineering. Instead of breaking through perimeter defenses directly, adversaries target upstream software vendors, open-source repositories, developer tools, or CI/CD build environments. By inserting malicious code into legitimate dependencies or automated software updates, attackers achieve massive blast radiuses across downstream organizations.

Industry reports link the surge in supply chain incidents to heavy reliance on external package registries and automated build pipelines (Verizon, 2025). To combat these systemic risks, researchers recommend enforcing Software Bills of Materials (SBOMs), strict cryptographic dependency verification, automated vulnerability checks, and code signing throughout development pipelines.

2.5.4 Artificial Intelligence-Assisted Attacks
Artificial intelligence is accelerating both offensive tooling and defensive monitoring. Adversaries leverage machine learning models to automate target scanning, craft targeted spear-phishing content, execute rapid credential stuffing, and generate evasive malware variants. Conversely, defensive teams deploy AI models for real-time log correlation, anomaly detection, and automated threat response. The primary challenge for organizations is operational speed: automated defensive controls must operate at the same velocity as AI-driven attack tools.

2.5.5 Synthesis of Emerging Threats

Literature on emerging threats confirms a clear trend: modern exploits easily cross traditional application borders. A single defect in an API endpoint, a cloud permission policy, or an open-source package can compromise an entire corporate software ecosystem. Effective security requires moving away from isolated tools toward integrated governance combining secure development lifecycles, cloud access management, dependency checks, and continuous runtime observability.

2.6 Modern Security Frameworks for Web Applications

Modern defensive frameworks seek to embed security validation early in development and maintain continuous monitoring after deployment. In distributed environments, security controls must cover source code, build pipelines, identity providers, and runtime infrastructure.

2.6.1 Secure Software Development Life Cycle (SSDLC)

The Secure Software Development Life Cycle (SSDLC) embeds security activities into every phase of software creation, replacing older approaches where testing occurred right before deployment. Core SSDLC practices include threat modeling, secure coding standards, Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), penetration testing, and continuous vulnerability management. Organizations using formal SSDLC models report fewer production bugs, faster bug resolution, and lower overall remediation costs. However, adoption studies emphasize that SSDLC programs require strong leadership backing, continuous developer training, and seamless integration into existing developer tooling.

2.6.2 Development, Security, and Operations (DevSecOps)

DevSecOps embeds automated security checks directly into Continuous Integration and Continuous Deployment (CI/CD) pipelines, establishing shared security responsibility across developers, security analysts, and operations teams. By automating security gates within build pipelines, DevSecOps catches bugs early without slowing release schedules. Implementation challenges remain common: team silos, security skills gaps in engineering teams, and friction when fitting automated tools into complex workflows. Thus, DevSecOps success relies as much on team collaboration and continuous training as it does on automated scanning tools.

2.6.3 Zero Trust Architecture

As organizations transition from perimeter networks to hybrid and multi-cloud environments, Zero Trust Architecture (ZTA) has become a primary defensive framework. Traditional perimeter models assumed implicit trust for any user or device inside the internal network. Zero Trust replaces implicit trust with continuous context-aware verification. Every access request—regardless of origin—must be authenticated, authorized, and validated before access is granted. Zero Trust is essential for securing web apps whose assets span multiple cloud providers and APIs. However, implementation demands significant investment in identity access management, policy enforcement, and real-time monitoring.

2.6.4 Synthesis of Modern Security Frameworks

SSDLC, DevSecOps, and Zero Trust solve different parts of the application defense puzzle. No single framework provides complete protection on its own. Resilient security programs combine secure software creation, automated build testing, continuous runtime observation, and identity-centric access control into an integrated operational framework.

2.7 Empirical Sector Analysis

Security breaches manifest differently across industry verticals, even though underlying technical gaps look identical on paper. Organizations face unique threat profiles depending on regulatory mandates, legacy technical debt, and business priorities.

Healthcare systems suffer constant targeting because health records command high premiums on dark web markets, while patient portals rely heavily on legacy backend infrastructure. Financial services deal with relentless automated attacks on web platforms and payment endpoints, where credential stuffing, session takeover, and API logic abuse are daily occurrences (Verizon, 2025). Educational institutions present wide attack surfaces due to open network access models, decentralized user bases, and chronic underfunding of IT security controls. Government portals face persistent state-sponsored probing that targets unpatched edge software and compromised supply

chain vendors. Across all four sectors, breach frequency correlates far more closely with delayed patch management and poor identity governance than with the choice of technology stack.

2.8 Synthesis of Literature

Looking at how application security research evolved, a clear pattern emerges. Academic and industry work began by focusing on individual code-level flaws—isolating bugs like SQLi, XSS, and CSRF within single, monolithic applications. That foundational work remains vital, but modern software stacks altered the rules. Third-party package registries, cloud identity platforms, microservice meshes, and AI-driven tooling created an interconnected risk landscape that perimeter firewalls cannot handle. Defensive strategies adjusted accordingly: the field moved toward continuous, lifecycle-wide governance models that combine secure coding practices, build-pipeline automation, strict identity verification, and runtime monitoring.

2.9 Research Gap

Despite extensive research covering specific exploit methods and defensive tools, critical gaps remain unaddressed. The vast majority of published studies analyze single vulnerabilities, standalone security tools, or specific industry sectors in isolation. Very little empirical work tracks how classic code-level defects interact with modern, distributed risks like API logic abuse, cloud IAM misconfigurations, supply chain poisoning, and AI-driven automation over long periods. Moreover, frameworks like SSDLC, DevSecOps, and Zero Trust are almost always studied as separate methodologies rather than as parts of a single, coordinated security program. This research addresses those gaps directly by pulling together empirical data across multiple sources to design an adaptive, unified defense framework for modern web applications.

Summary of Identified Research Gaps

Existing Research Focus	Identified Research Gap	Contribution of the Present Study
Focuses primarily on individual vulnerabilities such as SQL Injection, Cross-Site Scripting, and Cross-Site Request Forgery.	Limited synthesis of how web application attacks have evolved over time.	Provides a systematic analysis of the evolution of web application attacks from traditional vulnerabilities to emerging threats.
Examines traditional web application security threats or specific emerging technologies independently.	Limited integration of cloud-native computing, APIs, microservices, software supply chains, and AI within a single analytical framework.	Integrates traditional and emerging threats to present a holistic view of the contemporary web application threat landscape.
Evaluates defensive strategies such as SSDLC, DevSecOps, and Zero Trust independently.	Limited comparative synthesis of complementary modern security frameworks.	Examines how multiple security frameworks collectively contribute to strengthening web application security.
Investigates cybersecurity within specific industries or technology domains.	Limited cross-domain synthesis of empirical evidence across organizational contexts.	Synthesizes findings from diverse sectors to identify recurring trends, shared vulnerabilities, and common cybersecurity challenges.
Existing reviews lag behind the changing threat landscape.	Limited synthesis of AI-assisted attacks, supply-chain risks, API governance, and cloud security.	Provides an updated systematic synthesis of current web application security challenges.

III. RESEARCH METHODOLOGY

This chapter details the exact process we established to investigate how web application attacks have

changed over time and how we built a modernized defense framework around those findings. Given the applied, problem-solving nature of Information Technology research, we selected Design Science Research (DSR) paired with an Applied Mixed-Methods Threat Analysis Strategy. By combining academic papers with actual industry threat metrics, this hybrid approach allowed us to produce actionable recommendations tailored specifically for modern web application environments.

3.1 Research Framework

We anchored this study around the Design Science Research Methodology (DSRM) framework. We deliberately chose DSRM because simply describing security threats isn't enough; IT research needs to output working solutions. Through DSRM, we analyzed real-world security gaps and crafted a practical artifact—specifically, an Adaptive Web Application Security Framework and Mitigation Matrix—tailored for modern, cloud-native environments.

Our workflow moved through four distinct phases:

Phase 1: Problem Identification and Analysis

Extracting and organizing secondary data from incident reports and vulnerability databases.

Phase 2: Secondary Data Collection

Mapping older, established web attack types against emerging, API-centric threat vectors.

Phase 3: Comparative Threat Modeling

In phase three, we employ the STRIDE Threat Modeling Model to compare old web application designs with modern cloud-native and API-driven settings. This analysis aims to expose differences in attack surfaces, exploitation techniques, common vulnerabilities, and preventive mechanisms that have occurred in the evolution of web technologies.

Phase 4: Artifact Development

Building a modern security architecture alongside an adaptable DevSecOps assessment model

Phase 5: Dataset-Based Validation

Testing the framework against real exploit chains and having industry experts review its overall utility.

Figure 3.1 illustrates the research framework adopted in this study.

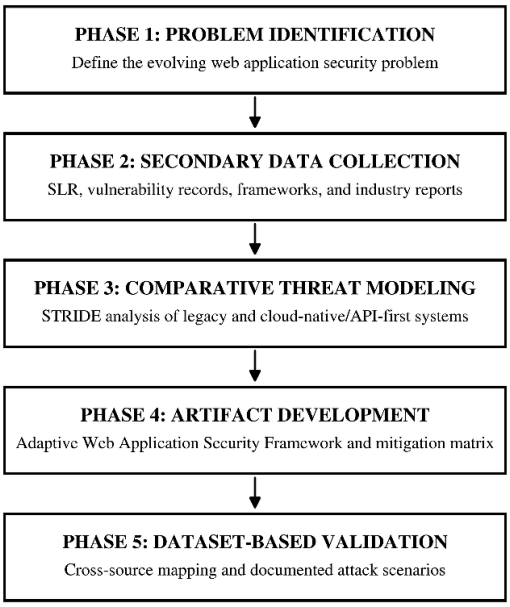

Figure 3.1. Design Science Research Process Adopted in the Study

3.2 Secondary Data Collection Strategy

To capture long-term statistical trends alongside real-world operational realities, we relied on a dual-tier sourcing strategy that combined peer-reviewed literature with trusted IT security data repositories:

3.2.1 Primary Data Sources & Threat Repositories

- Vulnerability Repositories: National Vulnerability Database (NVD) entries, Common Vulnerabilities and Exposures (CVE) records, and CISA's Known Exploited Vulnerabilities (KEV) catalog.
- Security Frameworks: OWASP Top 10 lists (spanning releases from 2013 up to present updates), the OWASP API Security Top 10, and application-layer tactics from the MITRE ATT&CK for Enterprise framework.
- Threat Intelligence Datasets: Verified telemetry reports published by major infrastructure and security vendors (including Cloudflare, Akamai, and the Verizon DBIR) tracking attack traffic volume, bot networks, and API exploitation patterns.

© AUG 2026 | IRE Journals | Volume 10 Issue 2 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I2-1722336

- Scholarly Literature: Indexed papers retrieved from IEEE Xplore, ACM Digital Library, and Scopus focused on web application defense from 2015 through 2026.

3.2.2 Inclusion and Exclusion Criteria

Before pulling literature and threat data, we set strict boundaries so everyone on the team followed the same rules for choosing sources. If a paper or industry report didn't hit our exact requirements, it was left out. Table 3.1 details the exact filters we used.

Table 3.1. Summary of Secondary Data Sources Used in the Study

Data Source	Purpose	Data Collected
National Vulnerability Database (NVD)	Analyze vulnerability trends	CVE entries, CVSS scores, CWE classifications
Common Vulnerabilities and Exposures (CVE)	Identify documented vulnerabilities	Vulnerability descriptions and identifiers
CISA Known Exploited Vulnerabilities (KEV) Catalog	Identify actively exploited vulnerabilities	Exploited vulnerabilities and affected products
OWASP Top 10	Analyze critical web application risks	Security risk categories
OWASP API Security Top 10	Examine API-specific threats	API security vulnerabilities
MITRE ATT&CK Framework	Analyze attacker tactics and techniques	Tactics, techniques, and mitigations
Verizon DBIR	Examine real-world attack trends	Breach patterns and incident statistics
Cloudflare Threat Reports	Analyze web and API attacks	Bot activity, DDoS, API attacks
Akamai Threat Reports	Examine emerging attack trends	Web application and cloud threats
IEEE Xplore, ACM Digital Library, and Scopus	Establish theoretical foundation	Peer-reviewed research articles

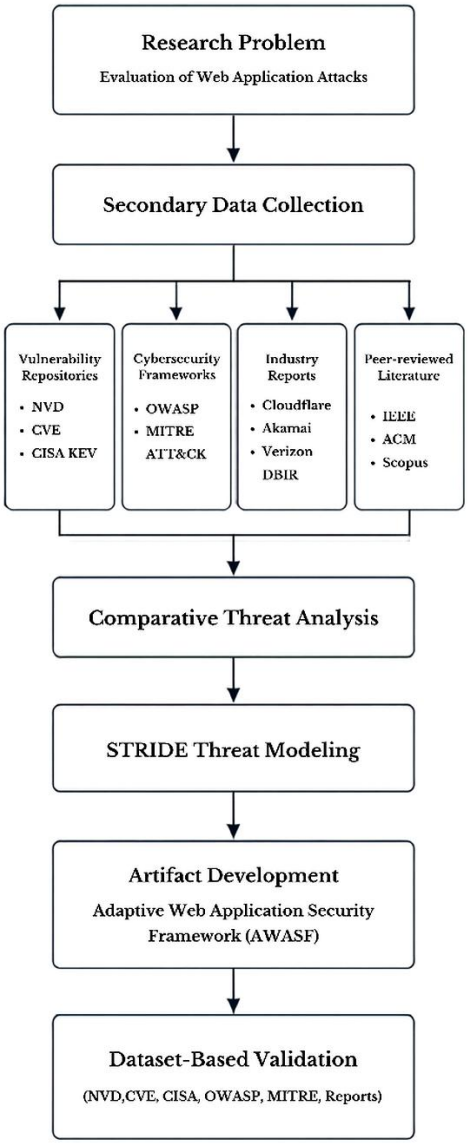

Figure 3.2. Research Workflow for the Development and Validation of the Adaptive Web Application Security Framework (AWASF)

© AUG 2026 | IRE Journals | Volume 10 Issue 2 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I2-1722336

Automated queries don't always work well for industry reports, so we manually pulled data from key threat intelligence platforms:

OWASP & MITRE Releases: We manually pulled official releases from specific milestone years. This included OWASP Top 10 releases (2013, 2017, and 2021), OWASP API Top 10 lists (2019 and 2023), and MITRE ATT&CK Enterprise matrices spanning versions 10 through 14.

Telemetry Data: We gathered annual security reports published between 2018 and 2026, focusing on concrete numbers for Layer 7 attack spikes, botnet activity trends, and API exploitation stats.

3.4 Systematic Literature Review Protocol

The academic literature was reviewed using a predefined protocol covering source selection, database searching, PRISMA screening, and quality assessment. The protocol creates an auditable record of how studies entered the final synthesis and supports replication of the review.

3.4.1 Inclusion and Exclusion Criteria

The criteria in Table 3.2 were applied during title-and-abstract screening and again during full-text assessment. Sources had to satisfy the scope, date, language, evidence, and relevance requirements to be retained.

Table 3.2. Inclusion and Exclusion Criteria for the Systematic Review

Category	Inclusion Criteria	Exclusion Criteria
Publication period	Peer-reviewed studies, security reports, and vulnerability statistics published from 2015 through 2026.	Materials published before 2015, except foundational models or historical benchmark editions.
Subject scope	Layer 7 vulnerabilities, web and API security, cloud-native architectures, attack evolution, and mitigation.	Studies limited to physical security, hardware, lower-layer networking, or client operating-system flaws.
Source quality	Indexed academic databases and official reports from recognized cybersecurity organizations.	Unverified forum posts, personal blogs, unsupported opinion pieces, and vendor marketing without empirical evidence.
Language and availability	Complete English-language texts with a stated method and accessible evidence.	Non-English publications, abstracts without full text, editorials, and duplicate records.
Research relevance	Sources reporting attack data, threat models, architectural analysis, or practical controls for web applications.	General security-awareness or software-engineering materials with no direct connection to web application security.

3.4.2 Database Search Strategy

We ran search strings across IEEE Xplore, ACM Digital Library, and Scopus to find our primary academic sources. Our team built these queries by combining three core themes—architecture type, threat style, and analytical focus—using standard operators like AND and OR, alongside wildcards where needed.

Table 3.3. Database Search Strings and Applied Filters

Database	Search String	Filters
IEEE Xplore	("Web Application*" OR "Microservices" OR "API Security") AND ("Threat*" OR "Vulnerability" OR "Attack Vector*") AND ("Evolution" OR "Trend*" OR "Comparative")	2015–2026; journals and conference publications; English
ACM Digital Library	+("web application" microservice "API security") +("threat model" vulnerability exploit) +("evolution" trend comparative)	2015–2026; research articles; English
Scopus	TITLE-ABS-KEY(("web application security" OR "API security" OR "cloud-native security") AND ("threat landscape" OR "attack evolution" OR "STRIDE") AND ("mitigation" OR "DevSecOps"))	2015–2026; articles and conference papers; Computer Science

3.4.3 Supplementary Search of Frameworks and Industry Reports

Automated queries don't always work well for industry reports, so we manually pulled data from key threat intelligence platforms:

- OWASP & MITRE Releases: We manually pulled official releases from specific milestone years. This included OWASP Top 10 releases (2013, 2017, and 2021), OWASP API Top 10 lists (2019 and 2023), and MITRE ATT&CK Enterprise matrices spanning versions 10 through 14.
- Telemetry Data: We gathered annual security reports published between 2018 and 2026, focusing on concrete numbers for Layer 7 attack spikes, botnet activity trends, and API exploitation stats.

3.4.4 PRISMA Screening and Study Selection

The selection process followed the PRISMA stages of identification, screening, eligibility, and inclusion. Searches identified 885 records. After 185 duplicates were removed, 700 unique records were screened. Title-and-abstract review excluded 512 records, leaving 188 full texts for eligibility assessment. A further 126 records were excluded after full-text review. The final synthesis contained 62 sources: 44 peer-reviewed studies and 18 framework or industry reports.

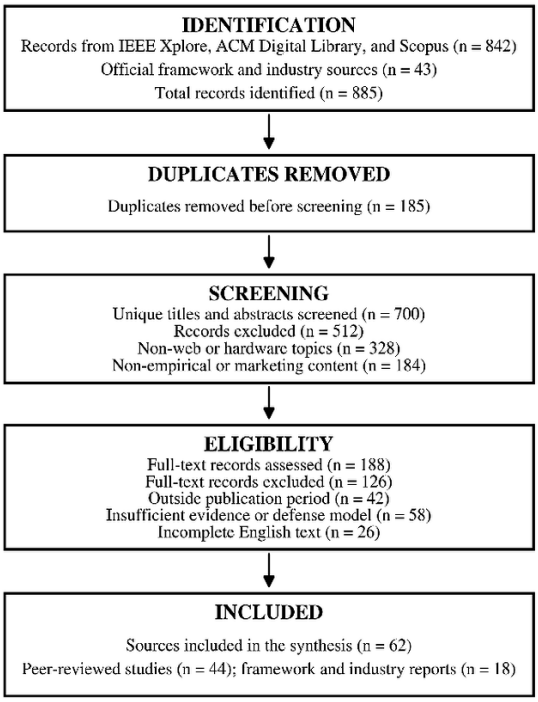

Figure 3.3. PRISMA flow diagram for study selection.

3.4.5 Quality Assessment of Selected Studies

We wanted to ensure that our framework rests on dependable data, so we ran every paper that passed full-text screening through a quick quality check using five specific questions.

Scoring System

© AUG 2026 | IRE Journals | Volume 10 Issue 2 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I2-1722336

We scored each paper using a simple 3-point scale:

- 1.0 point (Fully Met): The paper clearly hits the target and provides supporting proof.
- 0.5 points (Partially Met): It touches on the topic, but misses key details or testing depth.
- 0.0 points (Not Met): It doesn't address the point at all.

Out of a maximum 5.0 points:

- Studies scoring 3.5 or higher made up our primary evidence base.
- Studies scoring between 2.5 and 3.0 were used strictly for background context.
- Anything under 2.5 was thrown out because the data or methods were too weak.

Cut-off Score

Table 3.4. Quality Assessment Criteria for Selected Studies

ID	Quality Criterion	Assessment Question
QA1	Clear research scope	Does the source define the relevant Layer 7 architecture or attack vector?
QA2	Methodological transparency	Are its data collection and analysis procedures sufficiently clear to repeat?
QA3	Empirical support	Are its claims supported by verifiable records, telemetry, documented cases, or measurements?
QA4	Architectural evolution	Does it examine changes from legacy to cloud-native or API-first systems?
QA5	Practical relevance	Does it identify actionable controls or secure development recommendations?

3.5 Comparative Threat Analysis and Threat Modeling

To trace how attack vectors evolved over time (answering Research Questions RQ1 and RQ2), we adapted the STRIDE model (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) specifically for web architectures.

Table 3.5. Comparative Threat Analysis Dimensions

Architectural Dimension	Legacy Web Applications	Modern Web Applications
Primary Architecture	On-premise, monolithic architecture with server-side rendering	Cloud-native architecture utilizing microservices, containers, serverless computing, and single-page applications
Attack Surface	HTML forms, cookies, SQL queries, and URL parameters	REST and GraphQL APIs, OAuth tokens, CI/CD pipelines, containers, and third-party software packages
Dominant Vulnerabilities	SQL Injection (SQLi), Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and session hijacking	Broken Object Level Authorization (BOLA), software supply chain attacks, API abuse, and artificial intelligence-assisted exploits

Architectural Dimension	Legacy Web Applications	Modern Web Applications
Primary Defensive Mechanisms	Network firewalls, traditional Web Application Firewalls (WAFs), and manual security assessments	Web Application and API Protection (WAAP), Runtime Application Self-Protection (RASP), Zero Trust Architecture, and automated DevSecOps security practices

We directly compared two distinct operational eras across key application boundaries:

3.4 Artifact Development: Security Framework Design

Using our threat analysis as a foundation, we built the Adaptive Web Application Security Framework (AWASF). Designed for solution architects, software developers, and IT managers, the framework focuses on three core pillars:

1.) Shift-Left Pipeline Integration: Mapping actionable security gates directly into modern DevSecOps pipelines.
2.) Zero Trust Application Controls: Enforcing strict identity checks and explicit least-privilege boundaries between internal microservices and public APIs.
3.) Defensive Controls Matrix: A evaluation tool that balances security solutions (WAAP, RASP, AI-driven anomaly detectors) against setup complexity, operational overhead, and actual stopping power.

3.5.1 Legacy Web Applications

Legacy web applications are usually monolithic on-3.5 Framework Validation and Evaluation

To confirm that our methodology works as intended and that the resulting security framework actually holds up (addressing RQ3 and RQ4), we implemented a two-pronged evaluation strategy:

3.5.1 Scenario-Based Matrix Validation

We tested the framework against known historical and modern exploit chains—including major supply chain incidents like Log4j and complex API authorization exploits (BOLA). This helped us verify whether the proposed controls actually intercept an attack at various points along the kill chain.

3.5.2 Expert Evaluation / Rubric Scoring

We distributed a structured review questionnaire to a panel of experienced cybersecurity practitioners and IT leaders (including Senior Security Engineers, Solutions Architects, and DevOps Leads). The experts scored the framework on a 5-point Likert scale across four practical criteria:

- Comprehensiveness: How well the controls cover both legacy flaws and modern threat vectors.
- Practical Utility: How easily teams can insert these controls into real CI/CD pipelines.
- Scalability: Whether the framework scales effectively to cloud-native, containerized, and serverless environments.
- Clarity & Actionability: How easily technical leads can convert the framework guidance into daily engineering decisions.

3.6 Data Analysis & Synthesis Techniques

- Quantitative Trend Mapping: We ran descriptive statistical analyses on vulnerability databases, mapping category counts and severity shifts over time using CVE/NVD scores and movement within OWASP ranks.
- Qualitative Thematic Coding: We synthesized qualitative findings from research literature and technical advisories, organizing them into core security categories like API Authorization, Supply Chain Governance, and AI Threats.

3.7 Limitations and Ethical Considerations

Scope Boundaries: Our research looks strictly at Layer 7 (the application layer) and its immediate supporting web services. Hardware-level attacks, physical network exploits, and lower-level infrastructure vulnerabilities fall outside our focus.

Ethical Compliance: All testing relied purely on threat modeling models, public CVE entries, published incident breakdowns, and expert evaluation. We

© AUG 2026 | IRE Journals | Volume 10 Issue 2 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I2-1722336

performed no live penetration tests, unauthorized system scans, or real exploits, adhering fully to institutional research ethics policies.

IV. RESULTS AND DISCUSSION

This chapter reports the review's findings against the four research questions (RQ1–RQ4). The evidence is drawn from the empirical datasets and standards catalogued in the methodology: the OWASP Top 10 (2021) and OWASP API Security Top 10 (2023), the Verizon Data Breach Investigations Report, the Edgescan Vulnerability Statistics Report, and the NIST National Vulnerability Database together with the CISA Known Exploited Vulnerabilities catalog. Each section states what the data shows, interprets it against the reviewed literature, and connects it to the design of the proposed Adaptive Web Application Security Framework (AWASF). The chapter closes with a comparison against established frameworks and an honest account of the study's strengths and limitations.

4.1 Datasets Analyzed

This section reproduces the source datasets in full, so the figures behind the analysis are on the page rather than only cited. All figures are drawn from publicly available reports published within the study's five-year window (2021–2026). Four datasets were extracted: the OWASP Top 10 (2021) risk ranking with incidence data, the OWASP API Security Top 10 (2023) risk ranking, the Verizon DBIR breach-vector distribution, and the Edgescan and NVD/CISA KEV vulnerability metrics.

4.1.1 Dataset 1: OWASP Top 10 (2021) Web Application Risk Ranking

The OWASP Top 10 (2021) ranks the ten most critical web application security risks from data contributed across hundreds of thousands of applications. The list is reproduced below with the average incidence rate reported by OWASP for the categories where it was published (OWASP, 2021).

Table 4.1. OWASP Top 10 (2021) risk ranking. Incidence rates shown where published by OWASP; "—" indicates the rate was not reported for that category.

Rank	Risk Category	Change from 2017	Avg. Incidence
A01	Broken Access Control	Up from #5 (most CWE occurrences)	3.81%
A02	Cryptographic Failures	Up from #3 (renamed)	—
A03	Injection (now includes XSS)	Down from #1	3.37%
A04	Insecure Design	New category	—
A05	Security Misconfiguration	Up from #6	—
A06	Vulnerable and Outdated Components	Up from #9	—
A07	Identification and Authentication Failures	Down from #2	—
A08	Software and Data Integrity Failures	New category	—
A09	Security Logging and Monitoring Failures	Up from #10	—
A10	Server-Side Request Forgery (SSRF)	New category	—

4.1.2 Dataset 2: OWASP API Security Top 10 (2023) API Risk Ranking

The OWASP API Security Top 10 (2023) is a separate list for API-only services. Authorization and authentication flaws occupy the top positions, a structural contrast with the input-handling flaws that lead the general Top 10 (OWASP, 2023).

Table 4.2. OWASP API Security Top 10 (2023) risk ranking. Six of the ten categories concern authorization, identity, or dependency trust.

Rank	API Risk Category	Flaw Class
API1	Broken Object Level Authorization (BOLA)	Authorization
API2	Broken Authentication	Identity
API3	Broken Object Property Level Authorization	Authorization
API4	Unrestricted Resource Consumption	Availability
API5	Broken Function Level Authorization	Authorization
API6	Unrestricted Access to Sensitive Business Flows	Business logic
API7	Server-Side Request Forgery (SSRF)	Injection/trust
API8	Security Misconfiguration	Configuration
API9	Improper Inventory Management	Governance
API10	Unsafe Consumption of APIs	Dependency trust

4.1.3 Dataset 3: Verizon DBIR Breach Initial-Access Vectors

The Verizon 2025 DBIR analyzed 22,052 security incidents and 12,195 confirmed data breaches across 139 countries. The distribution below shows the leading initial-access vectors and the credential dependency of web application attacks (Verizon, 2025; Verizon, 2026).

Table 4.3. Verizon DBIR breach initial-access vectors and credential dependency (n = 22,052 incidents; 12,195 breaches).

Metric	Value	Edition
System intrusion (share of breaches)	53%	DBIR 2025
Social engineering	17%	DBIR 2025
Basic web application attacks	12%	DBIR 2025
Breaches beginning with credential abuse	22%	DBIR 2025
Breaches beginning with phishing	16%	DBIR 2025
Basic web app attacks involving stolen credentials	88%	DBIR 2025
Vulnerability exploitation as initial access (YoY change)	+34%	DBIR 2025
Exploited vulnerabilities in breach initial access	31%	DBIR 2026
Third-party involvement in breaches	30% (from 15%)	DBIR 2025

4.1.4 Dataset 4: Edgescan and NVD/CISA KEV Vulnerability Metrics

The Edgescan 2025 report draws on more than 40,000 assessments and 1,000+ penetration tests, cross-referenced with the NIST NVD and the CISA KEV catalog (Edgescan, 2025; NIST, 2021).

Table 4.4. Edgescan 2025 and NVD/CISA KEV vulnerability metrics.

Metric	Value	Source
CVEs published in 2025 (record)	48,185	Edgescan/ NVD
CISA KEV catalog entries (end 2025)	1,484 (+246)	Edgescan/ CISA

Metric	Value	Source
Most common critical web vuln (since 2022)	SQLi (CWE-89)	Edgescan
Full-stack vulns rated critical/high	>33%	Edgescan
Avg. MTTR, high/critical app & API flaws	54.81 days	Edgescan
Avg. MTTR, device/network flaws	39 days	Edgescan
Large-enterprise vulns unresolved after 12 months	45.4%	Edgescan
Testing workload: web apps / APIs	32–33% / 20–21%	Edgescan

4.2 Prevailing Web Application Attack Trends (RQ1)

The first research question asked which attack techniques currently dominate the web application threat landscape. Two patterns stand out. Credential-based attacks remain the leading route into web applications. The Verizon 2025 DBIR reports that 88% of basic web application attacks involved stolen credentials, and that credential abuse was the single most common initial access vector across the full breach dataset (Verizon, 2025). The second pattern is that exploitation of known vulnerabilities is rising sharply. The same report records a 34% year-over-year increase in vulnerability-based breaches, and the 2026 edition finds exploited vulnerabilities involved in roughly 31% of initial access (Verizon, 2026).

At the code level, injection-class flaws have not disappeared. The Edgescan 2025 Vulnerability Statistics Report identifies SQL Injection (CWE-89) as the most common critical web application vulnerability, a position it has held since 2022 (Edgescan, 2025). This directly supports the literature reviewed in Chapter 2, which argued that traditional flaws persist because of insecure implementation rather than a lack of awareness. The persistence is measurable: across the full stack, more than 33% of discovered vulnerabilities were rated critical or high severity, and critical application-layer issues took an average of 54.81 days to remediate (Edgescan, 2025).

Volume compounds the problem. The NVD recorded a record 48,185 CVEs published in 2025, and the CISA KEV catalog held 1,484 actively exploited entries by year-end, with 246 added during the year (Edgescan, 2025; NIST, 2021). The finding for RQ1 is therefore twofold: attackers still favour the easiest path—stolen credentials—while a widening backlog of unpatched, publicly known vulnerabilities give them a growing second front.

Table 4.5. Prevailing Attack Indicators Synthesized from the Datasets (2021–2026)

Indicator	Finding	Source
Credential-based web attacks	88% of basic web application attacks involved stolen credentials	Verizon (2025)
Vulnerability exploitation growth	34% year-over-year rise; ~20–31% of breach initial access	Verizon (2025, 2026)
Top critical web vulnerability	SQL Injection (CWE-89), most common since 2022	Edgescan (2025)
Disclosure volume	48,185 CVEs published in 2025 (record); 1,484 CISA KEV entries	Edgescan (2025); NVD
Application-layer remediation speed	Average MTTR of 54.81 days for high/critical app & API flaws	Edgescan (2025)

4.3 Comparative Analysis: Traditional versus Modern Attacks (RQ2)

The second research question asked how the modern attack surface differs from the legacy one. Applying the STRIDE model across the two operational eras defined in Chapter 3, the attack surface has moved outward from the monolithic application into the interfaces, identities, and dependencies that surround it.

© AUG 2026 | IRE Journals | Volume 10 Issue 2 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV10I2-1722336

Legacy monolithic applications concentrated risk in a small number of server-side entry points—HTML forms, cookies, URL parameters, and database queries—where SQL Injection, XSS, and CSRF dominated. Modern cloud-native applications distribute that risk. In the OWASP API Security Top 10 (2023), Broken Object Level Authorization (API1) and Broken Authentication (API2) are at the top of the API risk list showing a shift from flaws in input handling to flaws in authorization and identity (OWASP, 2023).

The Edgescan testing distribution supports the same reading. Its 2025 assessment workload split across network and cloud (about 47%), web applications (32–33%), and APIs (20–21%), confirming that APIs now make up a distinct and substantial share of the tested attack surface rather than an edge case (Edgescan, 2025). Authorization flaws, file path traversal, and injection persist as the highest-risk findings across that surface. The comparison below consolidates the shift.

Traditional flaws have not been replaced; they coexist with newer ones, which is why SQL Injection can top the critical-vulnerability list at the same time that BOLA tops the API list. And the newer flaws are structural rather than syntactic—they arise from how services trust each other and how identity is enforced, not from a single unescaped input. A defense model built only for the legacy surface will miss most of the modern one.

Table 4.6. Comparative Analysis of Traditional and Modern Web Application Attacks

Dimension	Legacy (Monolithic)	Modern (Cloud-Native / API-First)
Primary attack surface	HTML forms, cookies, URL parameters, SQL queries	REST/GraphQL APIs, OAuth tokens, CI/CD pipelines, third-party packages
Dominant vulnerabilities	SQLi, XSS, CSRF, session hijacking	BOLA/broken authorization, supply-chain compromise, AI-assisted exploitation
Ranking evidence	SQLi still top critical web flaw (Edgescan, 2025)	Broken Access Control #1 in OWASP Top 10 (2021); BOLA #1 in OWASP API Top 10 (2023)
Primary defense	Perimeter firewalls, traditional WAFs, manual audits	WAAP, Zero Trust, RASP, automated DevSecOps pipelines
Governing weakness	Insecure input handling	Authorization, identity, and dependency trust

Traditional flaws have not been replaced; they coexist with newer ones, which is why SQL Injection can top the critical-vulnerability list at the same time that BOLA tops the API list. And the newer flaws are structural rather than syntactic—they arise from how services trust each other and how identity is enforced, not from a single unescaped input. A defense model built only for the legacy surface will miss most of the modern one.*4.4 Emerging Security Challenges (RQ4)*

The fourth research question asked which emerging challenges most threaten current defenses. Three surfaced consistently across the sources.

4.4 Emerging Security Challenges (RQ4)
The fourth research question asked which emerging challenges most threaten current defenses. Three surfaced consistently across the sources.

4.4.1 Software Supply-Chain Exposure
Modern applications inherit most of their code from dependencies they did not write. This makes third-

party integrity a first-order security concern rather than a background one, matching the elevation of software integrity failures in the OWASP Top 10 (2021). The reviewed literature on the Log4j and comparable incidents shows how a single compromised component propagates to thousands of downstream systems, which is why the AWASF treats Software Bill of Materials (SBOM) generation and dependency scanning as pipeline requirements rather than optional add-ons.

4.4.2 AI-Assisted Offense and Defense

The 2026 DBIR confirms that threat actors are now abusing AI within their operations, while defenders apply it to detection and triage (Verizon, 2026). AI lowers the cost of reconnaissance, phishing generation, and credential-stuffing optimization, which increases the volume and speed of the credential attacks already dominating RQ1. This is an accelerant on an existing problem, and the framework responds with adaptive detection rather than static signatures.

4.4.3 The Widening Remediation Gap

The sharpest of these is the gap between disclosure and remediation. With a record CVE volume and an application-layer MTTR of roughly 55 days—and larger enterprises leaving about 45% of discovered vulnerabilities unresolved after twelve months (Edgescan, 2025)—the exposure window is expanding faster than most organizations can close it. Because Verizon (2025) reports an average time-to-exploitation of about five days for edge-facing flaws, the arithmetic is unforgiving: attackers routinely reach a vulnerability weeks before defenders patch it.

4.5 Implications for the Adaptive Web Application Security Framework (RQ3)

The third research question asked how effective current defensive controls are and how they should be combined. The findings above shape the AWASF in three concrete ways.

First, because credential abuse and broken authorization dominate, the framework centers on Zero Trust identity enforcement—continuous verification and least-privilege access across microservices and APIs—rather than perimeter defense. Verizon's finding that enabling MFA would have prevented major incidents such as the Snowflake breach (Verizon, 2025) makes identity the highest-leverage control in the matrix, consistent with NIST SP 800-207 (NIST, 2020).

Second, because the remediation gap is the binding constraint, the AWASF pushes controls left into the CI/CD pipeline—SBOM generation, dependency scanning, static and dynamic testing—so that vulnerabilities are caught before deployment rather than patched after exploitation. This directly targets the five-day exploitation window identified in RQ4.

Third, because traditional WAFs are documented as insufficient for API-specific authorization flaws, the framework specifies a defensive controls selection matrix that pairs each threat class with an appropriate layered control—WAAP and RASP for runtime, automated pipeline gates for build-time, and Zero Trust for identity—evaluated on deployment complexity, performance overhead, and mitigation efficacy. The intent is a control set matched to the modern attack surface rather than retrofitted from the legacy one.

4.6 Comparison with Related Security Frameworks

The AWASF does not replace established frameworks; it composes them into a single application-layer roadmap and adds the adaptive, evidence-driven prioritization that the individual frameworks leave to the practitioner. The table below positions it against the models most cited in the literature.

Table 4.7. Positioning of the AWASF Against Established Frameworks

Framework	Primary strength	Gap the AWASF addresses
OWASP Top 10 / API Top 10	Authoritative ranking of prevalent risks	Descriptive, not prescriptive; no pipeline integration guidance

Framework	Primary strength	Gap the AWASF addresses
NIST SP 800-207 (Zero Trust)	Rigorous identity and access model	Architecture-level; not mapped to web/API attack classes
MITRE ATT&CK	Evidence-based catalog of adversary tactics and techniques	Describes attacker behavior but does not prescribe application lifecycle controls or implementation priorities
SSDLC	Security across the development lifecycle	Process-focused; lacks runtime and supply-chain controls
DevSecOps	Continuous, automated security in CI/CD	Cultural/operational; no threat-to-control mapping matrix
AWASF (this study)	Integrates the above into one application-layer roadmap with a threat-to-control matrix	—

The comparison shows why integration is the central contribution. Each source framework is strong within its own scope, but practitioners still have to connect risk ranking, architecture, development practice, and runtime protection. AWASF makes those links explicit and uses the attack patterns identified in RQ1 and RQ2 to direct limited remediation resources toward the areas where the evidence shows the greatest exposure.

4.7 Practical Implementation of AWASF

Organizations can implement AWASF in stages. The first step is to inventory applications, APIs, identities, dependencies, and sensitive data flows, then rank exposed components using OWASP risks, NVD and CISA KEV records, and observed attack patterns. Controls can then be placed where they are most effective: secure coding and automated tests in CI/CD, multi-factor authentication and least privilege at the identity layer, WAAP or RASP at runtime, and SBOM-based dependency monitoring across the software supply chain.

Implementation should begin with internet-facing and high-impact services rather than trying to deploy every control at once. Teams should assign an owner and remediation date to each gap, then review the matrix after major architectural changes, new KEV entries, or security incidents. Useful operational measures include MFA coverage, the percentage of APIs tested for object-level authorization, critical dependencies represented in an SBOM, mean time to remediate exploited vulnerabilities, and detection and response time.

4.8 Framework Validation: Scenario-Based and Coverage Analysis

Following the analytical procedure defined in Section 3.7.1, AWASF is evaluated against documented attack chains rather than through live exploitation. Three public incidents were selected because they represent different threat classes that the framework must address: supply-chain compromise, credential-based account takeover, and API authorization abuse. Each scenario traces where an AWASF control would interrupt the attack. The final coverage matrix then tests the framework across the OWASP web and API benchmarks.

4.8.1 Scenario 1: Supply-Chain Compromise (Log4Shell, CVE-2021-44228)

Log4Shell was disclosed in December 2021 and remained exploitable across industries for years afterward. The attack chain is well documented: an attacker submits a crafted string such as a JNDI lookup expression into any field the application logs—commonly an HTTP header—after which the vulnerable Log4j library resolves the expression, reaches out to an attacker-controlled LDAP or RMI server, and loads remote Java bytecode, yielding remote code execution (CrowdStrike, 2021). Mass internet-wide scanning and public proof-of-concept code appeared within hours of disclosure. Mapping this chain against the AWASF: the Software Bill of

Materials (SBOM) and dependency-scanning controls in the shift-left pipeline would flag the vulnerable Log4j version at build time, before deployment; WAAP filtering would detect the JNDI lookup pattern in inbound requests at runtime; and Zero Trust egress restriction would block the outbound LDAP/RMI callback that the exploit depends on. The framework intercepts the chain at three separate stages, and the earliest—dependency scanning—prevents the vulnerable component from reaching production at all.

4.8.2 Scenario 2: Credential-Based Account Takeover (Snowflake, 2024)

The 2024 Snowflake-customer breaches affected roughly 165 organizations. Investigations by Mandiant and others confirmed that Snowflake's own platform was not compromised; instead, threat actor UNC5537 authenticated to customer tenants using valid credentials harvested by infostealer malware from employee devices, some dating back years (Cloud Security Alliance, 2025). The single decisive factor was that the affected accounts did not enforce multi-factor authentication, so a stolen username and password were sufficient; the accounts also lacked network allow-lists. Mapping against the AWASF: mandatory MFA under the Zero Trust identity layer would have blocked authentication with credentials alone; network allow-listing (least-privilege access) would have rejected logins from untrusted exit nodes; and continuous behavioral monitoring would have flagged anomalous bulk data access. This scenario is important because it involves no software vulnerability—the failure is organizational, which is exactly why the framework treats identity enforcement as its highest-leverage control rather than relying on patching.

4.8.3 Scenario 3: API Authorization Abuse (Dell Partner Portal, 2024)

In 2024, an attacker scraped about 49 million customer records from a Dell partner-portal API. The attacker registered a partner account—granted within about two days without verification—then queried an endpoint that returned customer details for a supplied seven-character service tag. Because the endpoint applied no authorization check on whether the partner was entitled to a given record, and the portal enforced no rate limiting, the attacker generated roughly 5,000 requests per minute for three weeks and enumerated records at scale undetected (Abrams, 2024). This maps directly to OWASP API1 (BOLA) and API4 (Unrestricted Resource Consumption). Mapping against the AWASF: object-level authorization checks would verify the requesting partner's entitlement to each record, defeating the enumeration; WAAP rate limiting would throttle the 5,000-requests-per-minute pattern; and API inventory and anomaly monitoring would surface the sustained bulk-lookup activity long before three weeks elapsed. The framework interrupts the chain at both the authorization and the volumetric stage.

4.8.4 Coverage Matrix

The scenario walkthroughs show interception of specific chains. To assess breadth, the AWASF's control domains are mapped against the OWASP Top 10 (2021) and OWASP API Security Top 10 (2023) categories established in Section 4.1. The matrix below records the primary control domain responsible for each category.

Table 4.8. AWASF coverage against OWASP Top 10 (2021) and OWASP API Security Top 10 (2023). "Direct" indicates a dedicated control domain; "Partial" indicates coverage that depends on correct configuration or complementary controls.

OWASP Risk Category	Primary AWASF Control Domain	Coverage
A01/API1/API5 – Broken/Object/Function Auth.	Zero Trust authorization enforcement	Direct
A02 – Cryptographic Failures	Secure-design + pipeline config checks	Partial
A03/API7 – Injection / SSRF	WAAP + RASP runtime filtering	Direct
A04 – Insecure Design	Shift-left threat modeling	Direct

OWASP Risk Category	Primary AWASF Control Domain	Coverage
A05/API8 – Security Misconfiguration	DevSecOps pipeline config gates	Direct
A06/API10 – Vulnerable Components / Unsafe Consumption	SBOM + dependency scanning	Direct
A07/API2 – Identification & Authentication	Zero Trust identity (MFA)	Direct
A08 – Software & Data Integrity	Code signing + integrity monitoring	Direct
A09 – Logging & Monitoring Failures	Continuous monitoring layer	Direct
A10 – Server-Side Request Forgery	Zero Trust egress restriction	Direct
API3 – Broken Object Property Level Auth.	Zero Trust authorization enforcement	Direct
API4 – Unrestricted Resource Consumption	WAAP rate limiting	Direct
API6 – Sensitive Business Flow Access	Behavioral monitoring + rate control	Partial
API9 – Improper Inventory Management	API inventory / discovery control	Direct

Applying the mapping rubric to 14 distinct OWASP Top 10 (2021) and OWASP API Security Top 10 (2023) categories produced 12 direct mappings and two partial mappings. Direct coverage is therefore 85.7%, while overall coverage is 100% because every category maps to at least one AWASF control. Using the weighted rubric defined in Section 3.7.1, the score is $(12 \times 1.0 + 2 \times 0.5) \div 14 \times 100 = 92.9\%$. The two partial cases, cryptographic implementation (A02) and sensitive business-flow abuse (API6), depend on correct configuration or supporting controls. The three scenario walkthroughs show that the mappings also hold against documented attack chains. These results answer RQ3 while locating the framework's main residual risk in implementation quality rather than missing control domains.

Table 4.9. Quantitative AWASF Validation Results

Validation measure	Result	Interpretation
Benchmark categories assessed	14	Combined distinct categories from OWASP Top 10 and OWASP API Security Top 10
Direct mappings	12 (85.7%)	A dedicated AWASF control domain addresses the category
Partial mappings	2 (14.3%)	Coverage depends on configuration or complementary controls
Unmapped categories	0 (0%)	Every assessed category maps to at least one AWASF control
Weighted coverage score	13.0/14 (92.9%)	Calculated with Direct = 1.0, Partial = 0.5, and None = 0
Documented scenarios intercepted	3/3 (100%)	AWASF controls interrupt the Log4Shell, Snowflake, and Dell API attack chains

This analytical validation has a clear boundary. It demonstrates that the framework's controls map onto and would intercept documented attack paths; it does not measure real-world deployment performance, operational overhead, or practitioner usability, which

would require field implementation or an expert panel. That limitation is stated explicitly in Section 4.8.

4.9 Strengths and Limitations

Strengths

The analysis rests on current, publicly verifiable datasets from recognized sources—OWASP, Verizon, Edgescan, and NIST—rather than on a single vendor's telemetry, which reduces source bias and lets any reader check the figures. Restricting the evidence to a five-year window (2021–2026) keeps the findings aligned with the present threat landscape rather than historical baselines. The study's main original contribution is the integration itself: it maps empirical attack frequencies onto a combined control framework, which the individual source frameworks do not do.

Limitations

The study is limited to the application layer (Layer 7) and its connected services; low-level network and hardware exploits are out of scope, so the picture is deliberately partial. The empirical datasets, while authoritative, reflect the visibility of the organizations that contribute to them and may under-represent sectors or regions that report less. The framework was validated analytically—through scenario-based mapping against documented exploit chains and coverage analysis against the OWASP benchmarks (Section 4.8)—rather than through live deployment. This method establishes that the controls would intercept known attack paths, but it does not measure operational overhead, deployment performance, or practitioner usability in production. Those dimensions would require field implementation or a structured expert-practitioner evaluation, which the study identifies as the natural next step for future work.

4.10 Summary of Findings

Across the four research questions, the evidence points in one direction. The dominant threats are credential abuse and a fast-growing backlog of exploitable vulnerabilities (RQ1); the attack surface has shifted from input handling in monolithic apps to authorization, identity, and dependency trust in cloud-native ones, while older flaws persist alongside the new (RQ2); the sharpest emerging challenge is a remediation gap that leaves vulnerabilities exposed for weeks against a five-day exploitation clock (RQ4); and the most effective response combines Zero Trust identity, shift-left pipeline controls, and layered runtime defenses into a single matrix (RQ3). The AWASF operationalizes that combination, and the scenario-based validation and coverage analysis in Section 4.8 demonstrate that it intercepts the documented attack chains across all three major threat classes.

V. CONCLUSION

The findings show that web application security now involves two connected problems. Long-standing vulnerabilities such as SQL Injection, Cross-Site Scripting, and Cross-Site Request Forgery remain common when secure coding, remediation, and configuration practices are applied inconsistently. At the same time, APIs, cloud infrastructure, identity services, software dependencies, and AI-enabled components have expanded the paths through which an attack can develop. Security programs must therefore maintain established application controls while managing risks across the wider system in which an application operates.

The study's main contribution is the Adaptive Web Application Security Framework (AWASF). Rather than proposing another stand-alone standard, AWASF brings together complementary principles from SSDLC, DevSecOps, Zero Trust Architecture, WAAP, and RASP within one adaptive structure. This integration links safe development with identity-based access control, runtime protection, dependency oversight, and continuous monitoring. It also provides a mechanism to address legacy weaknesses and emergent threats without separating them as security problems.

Theoretically, the work broadens the discussion of web application security beyond individual vulnerabilities and isolated protection techniques. It sees security as a consideration at every step of the lifecycle, influenced by the interactions between application code, infrastructure, identities and third-party components. It also accounts for controls that are technically sound but fail because they are implemented unevenly or without coordination. It also describes adaptive security as a strategy in which

multiple established controls work together and adjust to changes in the threat environment.

In practice, developers, security teams, organizations and educators can use the AWASF to analyze controls across development, deployment, and operation. It can help them identify gaps from policy to practice, especially for safe coding, access control, API governance, dependency management and monitoring. The framework has so far been tested against authoritative secondary evidence; the next stage is empirical testing in operational settings. Future research should consider applicability to businesses of different sizes, industries and security maturity levels, quantify changes in exposure, detection, response and remediation of vulnerabilities, and evaluate how the framework should evolve with new API, supply chain, cloud, and AI-assisted threats.

VI. RECOMMENDATIONS

6.1 Practical Recommendations

The findings suggest that organizations should treat web application security as a continuous responsibility rather than a perimeter to be guarded. Security belongs at every stage of the software life cycle, from secure design and implementation through deployment, monitoring, and maintenance. Embedding DevSecOps practices across that lifecycle reduces the cost of later remediation and strengthens the overall security posture.

Software developers should prioritize secure coding, code review, input validation, strong authentication and authorization, and timely remediation of known vulnerabilities. As API risks increase, teams should implement object-level authorization and least-privilege access, and then test these controls throughout development.

Cybersecurity specialists should supplement traditional Web Application Firewalls with identity-aware controls such as Zero Trust Architecture, continuous authentication, Runtime Application Self-Protection, and Web Application and API Protection solutions.

Organizations should also improve the security of their software supply chain by adopting Software Bills of Materials (SBOMs), dependency scanning, code signing, and regularly monitoring third-party software components throughout the software development lifecycle.

Schools, colleges, and universities that offer Information Technology and Computer Science programs must update their cybersecurity curricula to keep pace with the emerging threats in APIs, cloud-native computing, software supply chains, artificial intelligence, and today's secure software development practices. Using current cybersecurity frameworks and case studies in the classroom helps prepare future professionals for a fast-changing threat environment.

Organizations developing cybersecurity policies and governance frameworks should support globally accepted security standards and methods for continuous security assessment of traditional vulnerabilities and new threats in distributed software ecosystems.

6.2 Recommendations for Future Research

Future studies should include experimental validation of the proposed Adaptive Web Application Security Framework in operational environments to assess its performance across diverse organizational settings, software architectures, and industry sectors. These evaluations can further provide evidence of the possibility of adopting, scaling, and integrating the framework into existing software development and cybersecurity processes.

Future work can also explore the applicability of the framework to emerging computing environments such as serverless computing, edge computing, Internet of Things (IoT) ecosystems, and multi-cloud infrastructures where evolving technologies add new security challenges beyond those studied in this work.

Future studies must examine the long-term effects of AI-assisted attacks and AI-enabled security measures for adaptive web application protection techniques due to the rapid advancement of artificial intelligence in offensive and defensive cybersecurity. Research comparing current cybersecurity frameworks with artificial intelligence provides valuable information for improving automated threat detection and response capabilities.

Lastly, future systematic studies should expand their scope to include other cybersecurity datasets, industry threat intelligence feeds, and longitudinal research to track attack strategies that keep changing the web application threat landscape. These results would strengthen the body of data supporting robust, adaptable approaches to online application security.

ACKNOWLEDGMENT

First and foremost, we thank the Almighty God for providing us with the strength, wisdom, and determination to complete this study. Our faith in times of uncertainty gave us the courage to continue and complete our study.

We are especially grateful to Dr. Serafin Palmares for his guidance, expertise, patience, and useful feedback during the research process. His encouragement and support greatly helped us improve and complete our study.

We also want to express our gratitude to Dr. Kristine T. Soberano, MIT Program Head and Director for Research and Development, for her valuable guidance and support throughout our research. We also want to thank State University of Northern Negros (especially the Research Office) for their review of our study.

We also express our gratitude to our respective institutions and workplaces for their understanding and support as we balanced professional responsibilities with the demands of graduate studies and research. Their consideration and encouragement enabled us to pursue and complete this academic endeavor.

We also appreciate all members of our research group for their teamwork, commitment, and effort. Each one of you has been a treasure to work with, as your ideas, responsibilities, and support for each other have helped us overcome difficulties.

We also want to thank our colleagues, classmates, and friends for their suggestions, encouragement, and support while preparing this research.

Finally, we want to express our deepest gratitude to our families and loved ones for their understanding, patience, and support throughout this journey.

REFERENCES

[1] Abrams, L. (2024, May 10). *Dell API abused to steal 49 million customer records in data breach*. BleepingComputer. https://www.bleepingcomputer.com/news/security/dell-api-abused-to-steal-49-million-customer-records-in-data-breach/

[2] Cloud Security Alliance. (2025, May 7). *Unpacking the 2024 Snowflake data breach*. https://cloudsecurityalliance.org/blog/2025/05/07/unpacking-the-2024-snowflake-data-breach

[3] CrowdStrike. (2021, December 10). *Log4j2 vulnerability "Log4Shell" (CVE-2021-44228): Analysis and mitigation recommendations*. https://www.crowdstrike.com/en-us/blog/log4j2-vulnerability-analysis-and-mitigation-recommendations/

[4] Cybersecurity and Infrastructure Security Agency. (n.d.). *Known exploited vulnerabilities (KEV) catalog*. https://www.cisa.gov/known-exploited-vulnerabilities-catalog

[5] Edgescan. (2025). *Inside the 2025 Verizon DBIR: Critical insights on web application security*. https://www.edgescan.com/inside-the-2025-verizon-dbir/

[6] MITRE. (n.d.). MITRE ATT&CK. https://attack.mitre.org/

[7] National Institute of Standards and Technology. (2020). *Zero trust architecture* (NIST Special Publication 800-207). https://doi.org/10.6028/NIST.SP.800-207

[8] National Institute of Standards and Technology. (2021). *CVE-2021-44228 detail*. National Vulnerability Database. https://nvd.nist.gov/vuln/detail/CVE-2021-44228

[9] National Institute of Standards and Technology. (2024). *Cybersecurity Framework (CSF) 2.0*. https://www.nist.gov/cyberframework

[10] Open Worldwide Application Security Project. (2021). *OWASP Top 10: 2021*. https://owasp.org/Top10/

[11] Open Worldwide Application Security Project. (2023). *OWASP API Security Top 10: 2023*. https://owasp.org/API-Security/editions/2023/en/

[12] Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., & Moher, D. (2021). *The PRISMA 2020 statement: An updated guideline for reporting systematic reviews*. *The BMJ, 372*, Article n71. https://doi.org/10.1136/bmj.n71

[13] Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). *A design science research methodology for information systems research*. *Journal of Management Information Systems, 24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

[14] Verizon. (2025). *2025 data breach investigations report*. https://www.verizon.com/business/resources/reports/2025-dbir-data-breach-investigations-report.pdf

[15] Verizon. (2026). *2026 data breach investigations report*. https://www.verizon.com/business/resources/reports/dbir/