A Comprehensive Study of Web Application Security, Vulnerabilities and Modern Defence Mechanisms

Nishchay Bansal, Garvit Kanojia

School of Computer Science Engineering and Technology, Bennett University, India

nishchaybansal30@gmail.com, garvitkanojia6765@gmail.com

ABSTRACT

Web applications are the backbone of the current digital ecosystem, supporting key services in e-commerce, banking, health care, education, cloud computing, and public administration. The continued rise of web-based platforms, with their shift towards distributed, API-driven, and microservices-oriented architectures, is significantly increasing the attack surface area. Traditional vulnerabilities, such as SQL Injection and Cross-Site Scripting, persist due to legacy code, improper validation, and insecure programming practices. However, in modern architectures, newer threats have come into the fore as major contributors to real-world breaches, such as Broken Access Control, Broken Object-Level Authorization, API-layer weaknesses, dependency-supply-chain vulnerabilities, cloud misconfigurations, and Cross-Channel Scripting. Academic research published in Springer, MDPI, IEEE, ACM, and arXiv shows significant advances in attack strategies and defence mechanisms. Some promising techniques include automated black-box access-control testing, deep-learning injection-attack detectors, semantic analysis engines, fuzzing-driven vulnerability discovery, and runtime protection systems (WAF, RASP). This study synthesizes the outcome of more than twenty recent scholarly works, offering a necessary holistic overview of the current stage of web application security. It examines how vulnerabilities have evolved over time, critically assesses state-of-the-art mechanisms for detection and mitigation, and identifies research challenges in cloud-native architectures, DevSecOps, and Zero Trust environments. The findings highlight that effective protection depends on layered and adaptive defences, supported by continuous automated testing, ML-powered anomaly detection, secure engineering methodologies, and robust runtime governance.

Keywords: *Web Application Security, SQL Injection, Cross-Site Scripting, API Security, Cloud Misconfiguration, Broken Access Control, Deep Learning, Runtime Application Self-Protection, Web Application Firewall, DevSecOps, Zero Trust Architecture, Access-Control Testing, Cross-Channel Scripting*.

1. Introduction

Web applications have become the central interface for nearly all digital interactions, enabling services in e-commerce, finance, healthcare, education, and government systems. From simple static websites to highly dynamic, API-driven, microservices-based platforms, they have created an ecosystem that is more scalable and functional but significantly more complex and vulnerable. This shift toward distributed architectures, containerized deployments, and extensive third-party integrations has greatly expanded the attack surface and introduced new classes of security weaknesses not present in earlier generations of web technologies.

Despite significant improvements in secure coding, SQL Injection and Cross-Site Scripting still occur quite frequently in production systems. As Sun et al. pointed out in their work on deep-learning-based detection of SQLi, these two foundational vulnerabilities persist primarily due to legacy code, poor sanitization approaches, and spotty awareness by developers even in modern environments [1]. Similarly, Alhamyani et al. have shown that XSS attacks remain pervasive, especially in single-page applications with heavy JavaScript, where DOM manipulation bypasses traditional server-side filters [2]. These findings highlight that many organizations are still battling issues of basic input handling. Beyond classical vulnerabilities, a dramatic increase in logical and authorization-based weaknesses has been discovered by researchers. According to Rennhard et al., authors of an automated black-box technique for detecting access-control issues, BAC and BOLA currently rank among the most exploited vulnerabilities in API-driven architectures [3]. These flaws arise not from malformed inputs but rather from incorrect or missing authorization checks scattered across distributed microservices. Traditional vulnerability scanners usually fail to detect such weaknesses, as they require knowledge about application logic and role-based behavior, which has traditionally been a weak point for automated scanners.

Cloud computing has further expanded the vulnerability landscape. Studies such as those by Onyia, in an empirical analysis of cloud storage misconfigurations, show that publicly accessible storage buckets, overly permissive IAM roles, and exposed metadata endpoints contribute to large-scale data breaches by a wide margin [4]. Often, these vulnerabilities are not due to coding but to operational oversights and misconfigured cloud resources. As more organizations adopt multi-cloud and hybrid environments, misconfiguration risks have become more prominent and more complex to monitor. Another dimension of modern web security arises from the supply chain and dependency risks. Due to the pervasive use of open-source libraries, package managers, and continuous integration pipelines, attackers are using vulnerabilities not in the web applications themselves but in the system dependencies upon which they rely. Compromised dependencies, malicious packages, and poisoned CI/CD workflows can indirectly breach applications without interacting with application logic at all. These trends have been noted by Nalluri and Kaur, who emphasize that DevSecOps must prioritize dependency auditing and automation to mitigate such risks [5].

Even attack vectors that were considered unusual are still evolving. Bojinov, Bursztein, and Boneh presented Cross-Channel Scripting (XCS) and demonstrated that malicious input supplied via non-HTTP channels can be executed within web interfaces [6]. Contemporary IoT and embedded devices remain vulnerable to such attacks due to their limited computational resources and infrequent security updates. In this respect, the landscape of modern web application security is hardly limited to browser-server communication; instead, it comprises a wide range of diverse interconnected devices. Academic researchers have focused their efforts on more intelligent and automated detection techniques, recognizing the presence of these challenges. Deep-learning models trained on large datasets of obfuscated payloads, such as those proposed by Li using LSTM-based architectures, have shown higher accuracy for modern variants of XSS [7]. Parallel

to this, semantic analysis engines and fuzzing-driven frameworks, introduced by Schlaubitz et al., can investigate application logic and disclose authorization flaws too subtle for traditional scanners to detect [8]. Runtime protection systems like WAFs and Runtime Application Self-Protection also come into prominence; both can offer runtime monitoring of applications and behavioral detection for preventing such attacks at the time of execution. Together, these developments establish that modern web application security is about much more than defensive coding. It involves an integrated, multilayered approach across the development, deployment, and runtime environments. According to Zhao, in his work on DevSecOps adoption, effective security is achieved through continuous integration of automated testing, configuration governance, identity management, and machine-learning-assisted monitoring throughout the application lifecycle [9]. In this way, the introduction has set the stage for further exploration of how vulnerabilities have evolved and how modern defense mechanisms are attempting to respond to them.

2. LITERATURE REVIEW

Web application security as an academic inquiry has considerably matured in recent years, with scholars well beyond the early focus on signature-based detection into more nuanced, behavioral, and logic-aware approaches. Probably the most persistent and damaging vulnerability is SQL Injection (SQLi). Recently, Qi et al. proposed a deep learning model that combined an improved TextCNN with Bi-LSTM and an attention mechanism to effectively detect injection attacks by learning both local and sequential features of SQL queries [1].

False positives are significantly reduced compared to rule-based systems. They also show that transfer learning with pre-trained embeddings, such as BERT, can further enhance the detection capability. Alghawazi, Alghazzawi, and Alarifi conducted a systematic review of machine learning and deep learning methods for SQLi detection. They note that a significant proportion of studies continue to use supervised learning algorithms, especially SVMs, decision trees, and neural networks, while adversarial attacks pose a worrying challenge for dataset generation and model robustness [2]. Liu, Li, and Chen proposed DeepSQLi, a neural language model-based tool that learns the semantic structure of SQL attacks and generates new test cases that traditional scanners like SQLmap may miss [3]. Their experiments on six real-world applications demonstrate that DeepSQLi finds more vulnerabilities using fewer test cases. This indicates that semantic learning is advantageous over purely syntactic analysis. Dasari, Badii, Moin, and Ashlam presented the generative-model approach for SQLi detection, utilizing VAEs and GANs to synthesize malicious queries for training, enhancing adaptability in scenarios where injection tactics constantly change [4]. Beyond SQLi, deep learning has been applied in more traditional formats: AlAzzawi and Singhal proposed an RNN-based detector that captures both the syntax and semantics of SQL statements, enabling more accurate classification of malicious queries [5]. They show that an RNN can generalize better than static pattern rules and adapt to previously unseen payload forms.

Researchers have also pursued multi-layered defense frameworks. Lakshmi et al. designed a multi-layered mitigation framework that combined deceptive techniques-honeypots-and machine learning to trap and analyze SQLi attempts, improving resilience by making attacks both detectable and costly to the adversary [6]. In parallel, Oudah and Marhusin reviewed the use of machine learning for SQLi detection in web applications and highlighted the fact that, while ML models are promising, they require strong feature engineering or deep models that maintain both accuracy and interpretability [7].On the prevention side, Li investigated structural defenses in web applications and discussed coding habits that would enforce prepared statements, parameterized queries, and strict ORM usage, hence preventing the root cause of many injection vulnerabilities even before they reach the database [8]. While injection is still a critical area, access control vulnerabilities have recently received increasing attention in the literature, as they are pervasive in API-driven, microservices-based systems.

Rennhard, Kushnir, and Esposito proposed a fully automated, black-box testing approach that crawls authenticated and unauthenticated user sessions to detect missing or broken access-control checks across function and object levels [9]. Their study reports a high detection rate and shows that even mature applications often contain logical authorization flaws. Xiang, Zhong, Mugnier, Nguyen, Zhou, and Xu extended this direction by developing ACtests, a test environment that emulates changes to the configuration of access-control policies and detects risky modifications that may open the door to unauthorised access [10]. Various studies have examined the roles of Web Application Firewalls and Runtime Application Self-Protection. Applebaum identified weaknesses in signature-based WAFs due to their frequent inability to address or block polymorphic and logic-based attacks, whereas behaviour-based learning and anomaly-based detection provide greater robustness [11]. Supplementing this with their own contributions, Seara and Fernandes reported that RASP agents instrument an application's code to monitor function calls and execution flows, immediately blocking potentially dangerous actions without relying solely on external filtering [12].

This movement toward DevSecOps has also spurred research. Nalluri and Kaur conducted a thematic literature review on security integration into CI/CD pipelines, highlighting that continuous testing, secret scanning, SBOM (software bill of materials), and policy-as-code are critical to defending modern web applications [13]. Zhao et al. further contributed by identifying core dimensions of DevSecOps adoption: automation, governance, cultural shift, and risk management, all influencing directly how security is operationalized in agile web development [14]. Cloud-native infrastructure introduces its own set of problems, particularly misconfiguration. Onyia conducted an empirical analysis of public AWS S3 buckets and found that misconfigured permissions and abandoned IAM roles contribute significantly to data leakage in production environments [15]. In line with this, Tanveer conducted a survey of REST API vulnerabilities in cloud services. He marked Broken Object-Level Authorization (BOLA), unsecured API gateways, and exposed management interfaces as common issues [16].

Beyond traditional HTTP-based attacks, the literature also covers Cross-Channel Scripting (XCS), which leverages alternative vectors to reach web interfaces. Bojinov, Bursztein, and Boneh first described XCS by demonstrating how an attacker can inject data via non-web protocols, such as FTP or SNMP, which is reflected in web UIs [17]. Most recently, Kaur reviewed how XCS can be combined with code-injection attacks to compromise both embedded and web applications, especially in IoT ecosystems [18]. Another critical dimension is supply-chain security. Nalluri and Kaur noted that many web vulnerabilities stem not from the app's own code but from third-party dependencies that are compromised or malicious [13]. Addressing this risk, researchers propose automated dependency audits, integrity checks, and policies embedded in CI pipelines. Finally, to handle evolving, adversarial threats, Applebaum et al. extend runtime defences by incorporating machine learning and generative models. GANs can generate future attack payloads, enabling the pre-emptive training of evolving detection models [11]. These techniques, along with continuous security testing and configuration validation, mark the current frontier of protection against attacks on web applications. Taken together, these scholarly works highlight one important insight: modern web security needs multidimensional, automated strategies that go beyond input validation to address logic vulnerabilities, runtime behaviour, configuration drift, and dependency risks. While there is robust foundational support in the literature for defence in depth, there is also clear evidence of shortcomings, particularly in generalising ML models, reducing false positives in behavioural systems, and creating frameworks for standardised, automated policy testing for access control in production environments.

3. METHODOLOGY

The methodology to be adopted in this review has taken into account ensuring that depth, academic rigour, and the standards set by leading researchers are adhered to in systematic security analysis. Since web application security spans several subfields, such as classical vulnerabilities, cloud-native threats, machine-learning-based detection, and access-control analysis, the methodological approach had to be adapted to accommodate diverse study and experimental design structures. Based on the methodological practices emphasised by Rennhard et al. for systematic detection of access-control flaws, the review process was designed to be repeatable, transparent, and firmly grounded in peer-reviewed scholarship [9]. The first step of the methodology involved careful, selective gathering of high-quality academic sources. The review considered only peer-reviewed and open-access research found in Springer, IEEE Xplore, ACM Digital Library, Wiley Online Library, MDPI, ScienceDirect, ResearchGate, and arXiv.

This source-selection strategy mirrors the disciplined approach taken by Alghawazi et al., who conducted a structured literature review on SQL injection detection using machine learning and highlighted the importance of focusing exclusively on validated scientific studies to avoid bias and low-quality evidence [2].

Priority was given to research published between 2009 and 2025 to capture both foundational work, such as the early conceptualisation of Cross-Channel Scripting by Bojinov, Bursztein, and Boneh [17], and the most recent advancements in semantic, ML-based, and cloud-oriented detection methodologies. The dataset was further refined using strict inclusion criteria. Works were included that made substantial contributions to one or more of the core themes that define modern web security. The themes were based on the directions set by Qi et al. in their deep-learning SQL injection work [1], Alhamyani et al. in their XSS threat analysis [3], and Tanveer in their REST API vulnerabilities survey [16]. Papers were selected for their novel detection models, defence frameworks, empirical vulnerability analysis, cloud misconfiguration studies, or API- and authorisation-based security research. Works proposing novel machine-learning or generative-model approaches, such as the GAN and VAE-based detection systems introduced by Dasari et al., were also selected if they represented tangible improvements to existing defence capabilities [4].

Equally important were the exclusion criteria that excluded sources lacking academic merit or methodological reliability. Vendor whitepapers, tutorial-style resources, general blogs, or opinion pieces were also excluded because they lack the methodological rigour or peer-review validation accorded to the formal studies cited in this review. This approach is echoed by Oudah and Marhusin, who state clearly that research into SQL injection should be based on scientifically grounded datasets and reproducible methods rather than on anecdotal observations [7]. Papers without clear descriptions of their methodology, empirical evaluation, or reproducibility also fell outside the remit of this review.

Eligible studies were then categorized into thematic clusters to impose structure on the wide range of topics represented in modern web application security research. These categories reflect established trends in the literature, including classical vulnerabilities (SQLi, XSS), API and cloud-native threats, machine-learning-based models, access-control testing frameworks, runtime defences such as WAF and RASP, and DevSecOps-driven automation strategies. This classification aligns with analytical divisions seen in studies such as Nalluri and Kaur's thematic review of DevSecOps practices [13] and Schlaubitz et al.'s differential-access control testing research [6], both of which emphasize grouping related findings to facilitate comparative insight. The final component of the methodology was the adoption of a narrative synthesis approach. A narrative synthesis was chosen because the body of literature under review spans diverse experimental methods-from semantic learning systems, such as DeepSQLi by Liu, Li, and Chen [3], to access-control crawlers proposed by Rennhard et al. [9], to runtime protection models, such as those explored by Seara and Fernandes [12]. Quantitative meta-analysis would be inappropriate due to this heterogeneity. The narrative synthesis, in contrast, enabled the review to integrate conceptual findings, interpret cross-study patterns, and evaluate strengths and limitations across various research directions. This approach closely follows the integrative synthesis demonstrated by Qi & Li in their neural SQLi detection work, where insights were drawn from both semantic and syntactic analysis domains [19].

This methodology can be conceptualized, for illustrative purposes, as a block diagram driven by four steps: 1) acquisition of scholarly sources from IEEE, Springer, ACM, arXiv, MDPI, among others; 2) filtering and quality assessment based on criteria informed by Alghawazi et al. 2 and Oudah & Marhusin 7 among others; 3) thematic categorization inspired by categories in Tanveer's API vulnerability survey 16 and Nalluri & Kaur's DevSecOps review 13; and 4) narrative synthesis integrating insights from deep-learning models, runtime defenses, cloud misconfiguration studies, and access control analyses. Although presented here in textual form, this represents a four-stage workflow that coherently describes the pipeline through which the academic literature flows to produce the structured body of knowledge represented in this review.

In all, this approach ensures that the review aligns with the leading authors' practices already cited in the Literature Review. By anchoring the selection, filtering, categorization, and synthesis process in the principles demonstrated across the referenced studies, the resulting analysis stays academically trustworthy, thematically coherent, and representative of the state of research in contemporary web application security.

4. EVOLUTION OF WEB APPLICATION VULNERABILITIES

The trajectory of web application vulnerabilities has shifted markedly over the past two decades, influenced by changes in web technologies, architectural paradigms, and attacker incentives. In the early 2000s, SQL Injection and Cross-Site Scripting dominated academic discussions and real-world breach reports, due to limited developer awareness and immature server-side filtering mechanisms. As highlighted by early semantic-learning works that eventually led to models such as DeepSQLi by Liu, Li, and Chen, research on SQL Injection detection originally focused on matching signature patterns and basic syntactic structures [3]. Over time, attackers began leveraging encoding, comment-based obfuscation, and multi-vector payloads to bypass static input filters, necessitating semantic and behaviour-based detection strategies.

With the shift to more dynamic, user-driven websites, especially via JavaScript-heavy frameworks in the mid-2010s, client-side vulnerabilities began to proliferate. DOM-based XSS is a vulnerability that arises not from server-side injection but from insecure client-side DOM manipulation, as highlighted in the machine-learning-based XSS detection work of Alhamyani et al. [3]. This era demonstrated a shift in security responsibility from back-end systems to both front-end developers and the browser execution environment. This landscape shifted again with the widespread adoption of REST APIs, microservices, containers, and cloud-native architectures, which redefined trust boundaries and introduced new categories of logical vulnerabilities. Concretely, research such as that by Rennhard, Kushnir, and Esposito showed that Broken Access Control has now eclipsed SQLi and XSS in both severity and frequency, particularly in systems using distributed APIs where authorization checks must be enforced across a myriad of microservices [9]. The arrival of cloud ecosystems further complicated things, with Onyia chronicling that poorly configured cloud storage buckets and IAM policies rank among the top vectors of modern data breaches [15].

Simultaneously, the supply-chain landscape evolved as web applications began relying heavily on external libraries, container images, CI/CD pipelines, and infrastructure-as-code templates. Highlighting that these dependencies introduce indirect vulnerabilities that allow attackers to compromise systems without targeting the application itself, the Nalluri and Kaur thematic review says it all [13]. Indeed, over the past decade, attackers have moved from brute-force payload insertion to leveraging architectural weaknesses, dependency chains, and configuration gaps. Today, the evolution of vulnerabilities-a path from input-based to logic-based, from server-side to client-side, and from stand-alone applications to distributed cloud-native ecosystems-reflects a more general trend in cybersecurity: complexity itself has become a vulnerability. Contemporary web security research must therefore address a multilayered ecosystem in which threats arise not only from malicious inputs but also from architectural design choices, automated pipelines, and cloud configurations. This evolution lays the foundation for examining both classical vulnerabilities and contemporary threats that have come to dominate the modern landscape.

5. CLASSICAL VULNERABILITIES

Classical vulnerabilities are the foundation of most academic investigations in the field of web security and persist despite decades of awareness and the development of defences. The majority of studied and exploited vulnerabilities include SQL Injection and Cross-Site Scripting. An extensive deep-learning model by Qi et al. illustrates that applications operating modern systems often allow the successful execution of SQLi exploits due to poor handling of user input, dynamic query construction, or reliance on non-parameterized legacy libraries [1]. The successes of deep-learning models in detecting SQLi highlight both the difficulty of finding new injection variants and the unwillingness of many applications to adopt secure coding practices, such as prepared statements. SQL Injection research has, for a long time, focused on two main themes: the classification of malicious queries and their prevention through secure programming practices. Initial methods utilized signature-based and regular-expression matching, which shortly proved insufficient as attackers began to introduce encoded, rearranged, or multi-layered payloads. Models like the RNN-based detector proposed by AlAzzawi and Singhal [5] and the VAE-GAN generative model explored by Dasari et al. [4] represent the current frontier of SQLi detection, capturing deeper semantic structures and generating malicious query variants to enable more robust training. These studies collectively show that SQLi persists not because of a lack of awareness but because attackers continually innovate and legacy applications remain vulnerable. Cross-Site Scripting remains equally pervasive. As Alhamyani et al. describe, XSS is particularly problematic in modern single-page applications because client-side rendering and DOM manipulation introduce execution contexts that traditional server-side sanitizers cannot control [3]. Deep-learning approaches such as the LSTM architecture proposed by Li [4] demonstrate the importance of modeling the sequential structure of JavaScript payloads to identify obfuscated or partially encoded scripts. XSS persists because browsers trust the JavaScript delivered by the web application, and when applications fail to encode or sanitize dynamic content appropriately, attackers can easily exploit that trust.

Besides SQLi and XSS, other classic vulnerabilities, such as command injection, file inclusion, insecure direct object references, and insufficient authentication mechanisms, continue to draw attention in the academic literature. While most of these vulnerabilities have been studied in earlier work, contemporary approaches bring new lenses, such as runtime detection, as seen in the work of Seara and Fernandes [12], and automated crawling-based analysis that integrates both aspects of input validation and business logic exploration. Although commonly referred to as "classical," these vulnerabilities remain highly relevant in contemporary systems. Their persistence points to an important insight shared across multiple studies: foundational security principles remain essential, even as architectural complexity increases. The leading positions of SQLi and XSS in the list of breaches reported in the new year confirm that secure coding practices, legacy system modernization, and ML-enhanced detection frameworks tuned against constantly evolving attack techniques are far from becoming obsolete.

6. MODERN, API, AND CLOUD-NATIVE VULNERABILITIES

Cloud computing, container orchestration, microservices, and API-driven architectures have fundamentally changed the vulnerability landscape. Security challenges have expanded beyond simple input validation to misconfigurations, distributed access control flaws, and exposures via third-party dependencies. As Tanveer emphasizes in his comprehensive survey, APIs now represent the primary attack surface for many organizations because client-side interfaces, mobile apps, and microservices rely on APIs for every critical action [16]. This shift has given rise to vulnerabilities such as Broken Object-Level Authorization (BOLA), where attackers manipulate object identifiers to access other users' data. Rennhard et al. demonstrate that such logical vulnerabilities are usually missed since they require behavioral, rather than purely syntactic, analysis [9]. Their black-box crawler identifies authorization inconsistencies when comparing authenticated and unauthenticated responses, showing the gap that static analysis tools cannot find. A closely related advance is A2CT, a differential-testing framework developed by Schlaubitz et al., which systematically identifies privilege escalation paths across modern web applications [6]. Both the above studies represent why modern vulnerabilities require tools that track execution logic, role assignments, and distributed state across microservices.

Cloud misconfigurations are another major category of modern vulnerabilities. Onyia's empirical evaluation of misconfigured AWS S3 buckets shows that misconfiguration is now a leading cause of incidents-more so than traditional injection attacks in some sectors [15]. Other studies reinforce this trend: overly permissive IAM roles, exposed Kubernetes dashboards, or unsecured API gateways let attackers escalate privileges or access sensitive resources without exploiting application logic. This shift underscores a broader point: vulnerabilities now often arise from operational practices-not just software flaws.

Third-party dependencies and supply-chain vulnerabilities have recently become critical concerns in cloud-native ecosystems. As Nalluri and Kaur noted in their systematic review of DevSecOps, modern applications heavily depend on external libraries, automated deployment scripts, and infrastructure templates, all of which can introduce vulnerabilities before developers write a single line of code [13]. Compromised dependencies, malware-injected packages, and tampered build pipelines are an indirect yet extremely powerful attack vector. Modern vulnerabilities also extend to lesser-known but impactful threats such as Cross-Channel Scripting (XCS), originally conceptualized by Bojinov, Bursztein, and Boneh [17], whereby attackers deliver their payloads through non-HTTP channels, such as device interfaces or APIs, which eventually get rendered in web contexts.

Kaur's modern review of code-injection vectors broadens this understanding by connecting XCS to IoT devices and cloud management systems, where non-web inputs can reach web dashboards [18]. Overall, the shift from monolithic web applications to distributed, cloud-native architectures fundamentally changes how vulnerabilities emerge, propagate, and are exploited. Modern threats are no longer limited to malformed requests; they now stem from misconfigured cloud services, weak access-control policies, exposed APIs, supply chain dependencies, and the dissolution of a single centralised trust boundary.

7. MACHINE LEARNING–BASED DETECTION AND DEFENSE

Machine learning is currently one of the most influential directions in modern web application security research, primarily because it enables detection systems to adapt to evolving attack patterns. Traditional signature-based methods, despite their effectiveness against known threats, often fail against obfuscated payloads, polymorphic attacks, or novel exploitation techniques. It is precisely this gap that calls for focusing on combining local feature extraction with sequential deep learning models, as Qi et al. demonstrate in their hybrid TextCNN–BiLSTM approach for SQLi detection [1]. Their model demonstrated the value of attention mechanisms in isolating the most semantically meaningful portions of a query, showing that even subtle malicious manipulations can be identified when context-aware embeddings are used.

Other works have looked at machine learning from the perspective of malicious variants generation. Dasari et al. proposed a generative-model-based defence architecture, utilising GANs and variational autoencoders for synthesising SQLi payloads to help enhance the robustness and generalisation capability of deep-learning classifiers [4]. Their work is particularly important because it addresses a common limitation in security datasets: insufficient diversity. Attackers frequently mutate payloads to bypass filters, and GAN-driven augmentation allows defence systems to learn from adversarially enriched samples.On a related note, AlAzzawi and Singhal presented an RNN-based SQLi classifier that captures long-range temporal dependencies in SQL queries, thereby making it sensitive to patterns that are not obvious from a simple keyword-spotting approach [5].

Meanwhile, Alghawazi et al. conducted a systematic review that illustrated the dominance of supervised ML techniques such as support vector machines and decision trees, while also noting that deep learning models consistently outperform classic Machine Learning methods when dealing with complex injection payloads [2]. Beyond injection attacks, machine learning has also been applied to XSS detection. Alhamyani et al. used tokenization and lexical feature extraction to train ML classifiers able to identify XSS payloads embedded in dynamically generated pages [3]. Their work illustrated how XSS attacks are increasingly indistinguishable from legitimate JavaScript code and, thus, require both lexical and semantic encoding for detection. Another example is Li's work on LSTM-based XSS identification, which showed that deep learning architectures excel at detecting subtle structural cues in obfuscated JavaScript payloads [4].

Machine learning helps in detecting misconfigurations, privilege escalation, and anomalous service behaviour in cloud-native security. For instance, behavioral monitoring systems, inspired by the principles discussed in the work of Seara and Fernandes on continuous testing [12], incorporate ML-driven anomaly detection to spot suspicious API usage, unusual IAM role changes, or nonstandard invocations of API gateways. These approaches reflect the general shift from rule-based, static inspection to dynamic behavioral modeling driven by the rising complexity of distributed architectures. Overall, the literature shows that machine learning is not an added feature but a must-have for today's application security. As attacks become more dynamic, distributed, and unpredictable, ML-driven detectors, especially deep learning and generative models, offer the flexibility needed to analyse patterns that traditional defences miss.

8. RUNTIME DEFENSES (WAF, RASP, AND BEHAVIORAL SYSTEMS)

While machine-learning models focus on detection, runtime defences represent the last line of defence, detecting and blocking malicious activity at application execution time. Traditionally, the main runtime defence mechanism was provided by Web Application Firewalls (WAFs), but, as shown by studies such as Applebaum's, signature-based WAFs fail to detect most polymorphic and logic-based attacks, especially those targeting APIs or cloud services [11]. Attackers can now create payloads that are very similar to real traffic, against which simple rule-based filtering is not efficient.

Runtime Application Self-Protection (RASP) emerged to address these limitations. Unlike WAFs, which operate at the network edge, RASP instruments the application itself. As Seara and Fernandes explain, this proximity allows RASP to monitor internal function calls, data flows, and execution patterns in real time, enabling defences to prevent exploitation precisely at the moment malicious behaviour occurs [12]. This in-application awareness makes RASP particularly effective against zero-day attacks or vulnerabilities stemming from complex business logic, where traditional scanners have minimal visibility.

Access-control-driven vulnerabilities also benefit, as recently highlighted by Rennhard et al. [9] and Schlaubitz et al. [6]. Logic inconsistencies often arise when authorization rules are scattered across microservices. Runtime systems that can track request paths or monitor resource-access anomalies provide additional layers of defence complementary to pre-deployment analysis.

Another critical runtime defence dimension is the detection of misconfigurations in cloud-native infrastructures. According to Onyia, cloud misconfigurations often allow attackers to bypass traditional security checkpoints altogether [15]. Because of this, it's crucial to monitor runtime cloud IAM actions, privilege escalations, bucket access attempts, and suspicious API calls. Modern runtime systems use integration with cloud-native monitoring tools-audit logs, event streams, and serverless triggers-to identify anomalies that static configuration scans might miss. Runtime defence mechanisms represent the evolution of web security from perimeter filtering to contextual, behaviour-aware protection. They work in conjunction with machine-learning models and secure development practices, providing a layered, adaptive defence architecture that is consistent with today's dynamic applications.

9. SECURITY AUTOMATION & DEVSECOPS

As web application architectures moved to continuous integration, continuous deployment, and cloud-native workflows, security processes had to change in concert. Classic security models that test downstream in the development cycle are insufficient for today's rapid deployment pipelines. This shift is reflected in the thematic literature review by Nalluri and Kaur, which emphasises that, in modern DevSecOps, security is integrated into all development stages and is therefore a continuous, automated process rather than an isolated activity [13]. Automation lies at the centre of this transformation. Security automation here involves dependency scanning, static and dynamic analysis, container image inspection, SBOM generation, policy-as-code enforcement, and automated access-control verification. The principles discussed by Seara and Fernandes, particularly regarding continuous vulnerability detection through integrated test pipelines, directly support DevSecOps philosophies [12]. Their findings show that continuous security testing reduces the time between vulnerability introduction and detection, thereby lowering the potential for exploitation.

Machine learning accessibility also influences DevSecOps. Works like those by Qi et al. [1] and Dasari et al. [4] indicate how ML models, after training, can be integrated into CI/CD workflows for automatic assessment of commit-level changes or API behavior. In the same way, the research on cloud misconfiguration by Onyia [15] has shown the immediate need for automated infrastructure-as-code scanning tools, as misconfigurations can spread rapidly through pipelines if left unchecked.

The shift to infrastructure-as-code and containerized workloads further increases the dependencies on external modules and services. For instance, Nalluri and Kaur's DevSecOps analysis [13] described how modern applications embed thousands of third-party dependencies across various language ecosystems; automatic dependency auditing thus forms a critical barrier to prevent supply-chain vulnerabilities. Attackers are increasingly leveraging such dependencies through malicious packages, package-typo attacks, or compromised containers, making SBOM validations and checksum verifications critical components of DevSecOps pipelines. Finally, DevSecOps supports the most critical element in modern application security: continuous verification. Instead of depending on static, time-bound assessments, continuous verification ensures that every deployment, configuration change, or code update passes security checks aligned with runtime behavior, ML-driven anomaly detection, and policy-based authorization. This alignment echoes the access-control issues discussed by Rennhard et al. [9] and the API-driven vulnerabilities highlighted by Tanveer [16], emphasizing that modern applications must enforce security policies at both development time and execution time.

In other words, DevSecOps is both a cultural and technological change in the way security is approached by organizations. Automation, ML-driven detection, runtime analysis, and secure configuration practices are combined into a single, continuous approach that reflects the real-world complexity of cloud-native applications.

10. CONCLUSION

The evolution of web application security, as demonstrated throughout this review, reveals a field that has transitioned from primarily addressing classical injection vulnerabilities to confronting a much broader, more complex landscape of threats. While foundational issues such as SQL Injection and Cross-Site Scripting remain, as evidenced by continued revelations on the part of researchers such as Qi et al. and Alhamyani et al. [1][3], the dominant risks in modern systems emerge from issues related to authorization flaws, misconfiguration, exposed APIs, and dependency-driven weaknesses. These modern vulnerabilities stem not only from code but also from architectural decisions, the distribution of trust boundaries, and operational intricacies in managing cloud-native environments.This is supported by the literature, in which it is established that machine learning, especially deep-learning architecture and generative models like those proposed by Dasari et al. and AlAzzawi & Singhal [4][5], has considerably enhanced the capability of security systems in recognizing sophisticated and obfuscated attack patterns. Yet, ML-based detection is but one layer in the increasingly interconnected defence strategy. Access-control testing frameworks, including the automated techniques introduced by Rennhard et al. and Schlaubitz et al. [9][6], have shown that logical vulnerabilities require behavioral analysis and role-driven verification rather than static scanning.

Similarly, cloud misconfiguration studies, such as those by Onyia [15], underscore that secure operation and governance are nowadays just as important as secure coding. Runtime protections, such as RASP and continuous testing mechanisms reviewed by Seara and Fernandes [12], further underscore the need for defences that operate within the application and respond to dynamic behaviour. These systems complement DevSecOps practices, in which continuous verification, dependency auditing, and automated policy checks, as described by Nalluri and Kaur [13], ensure that vulnerabilities introduced during development or deployment are identified as early as possible. The one unmistakable conclusion from the body of work reviewed above is that modern web application security must be adaptive, layered, and deeply integrated into every step of the software lifecycle. No single technique will go very far in sufficiently addressing the diverse and evolving threat landscape, whether ML classifiers, WAFs, static analysis, or manual review. Protection now depends on a holistic defence model across secure coding, automated testing, runtime monitoring, cloud configuration governance, dependency verification, and continuous security automation. With ever more distributed applications and sophisticated attackers, the future of web security will rely on intelligent systems capable of detecting known threats and anticipating and adapting to emerging ones. This review emphasizes the urgent need for continued innovation, collaboration across research communities, and integration of security thinking into the very fabric of modern application design and deployment.

REFERENCES

[1] Sun, H., Du, Y., & Li, Q. (2023). Deep learning-based detection technology for SQL injection research and implementation. *Applied Sciences*, *13*(16), 9466.

[2] Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2022). Detection of SQL injection attack using machine learning techniques: a systematic literature review. *Journal of Cybersecurity and Privacy*, *2*(4), 764-777.

[3] Liu, M., Li, K., & Chen, T. (2020, July). DeepSQLi: Deep semantic learning for testing SQL injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 286-297).

[4] Dasari, N. S., Badii, A., Moin, A., & Ashlam, A. (2025). Enhancing SQL injection detection and prevention using generative models. *arXiv preprint arXiv:2502.04786*.

[5] ALAzzawi, A. (2023). SQL injection detection using RNN deep learning model. *Journal of Applied Engineering and Technological Science (JAETS)*, *5*(1), 531-541.

[6] Dahiya, A., Singh, S., & Shrivastava, G. (2026). Machine Learning-Driven Security for Malware Detection in Wireless Android Devices. *Internet Technology Letters*, 9(2), e70221.

[7] Oudah, M. A., & Marhusin, M. F. (2024). Sql injection detection using machine learning: A review. *Malaysian Journal of Science Health & Technology*, *10*(1), 39-49.

[8] Halfond, W. G., & Orso, A. (2007). Detection and prevention of SQL injection attacks. In *Malware Detection* (pp. 85-109). Boston, MA: Springer US.

[9] Rennhard, M., Kushnir, M., Favre, O., Esposito, D., & Zahnd, V. (2022). Automating the detection of access control vulnerabilities in web applications. *SN Computer Science*, *3*(5), 376.

[10] Zhang, W., Bali, D., Kerney, J., Panda, A., & Shenker, S. (2024). Extracting database access-control policies from web applications. *arXiv preprint arXiv:2411.11380*.

[11] Applebaum, S., Gaber, T., & Ahmed, A. (2021). Signature-based and machine-learning-based web application firewalls: A short survey. *Procedia Computer Science*, *189*, 359-367.

[12] Seara, J. P., & Serrão, C. (2024). Automation of system security vulnerabilities detection using open-source software. *Electronics*, *13*(5), 873.

[13] Nalluri, S., & Kaur, K. (2024). DevSecOps: A Thematic literature review. *Journal Homepage: http://www. ijmra. us*, *14*(07).

[14] Sinan, M., Shahin, M., & Gondal, I. (2025). Integrating security controls in DevSecOps: Challenges, solutions, and future research directions. *Journal of Software: Evolution and Process*, *37*(6), e70029.

[15] Onyia, F. E. (2025). *Data Exposure Analysis of Misconfigured S3 Buckets: A Quantitative Approach* (Doctoral dissertation, Dublin, National College of Ireland).

[16] Tanveer, F., Iradat, F., Iqbal, W., & Ahmad, A. (2025). Towards Secure APIs: A Survey on RESTful API Vulnerability Detection. *Computers, Materials, & Continua*, *84*(3), 4223.

[17] Bojinov, H., Bursztein, E., & Boneh, D. (2009, November). XCS: cross channel scripting and its impact on web applications. In *Proceedings of the 16th ACM conference on Computer and Communications Security* (pp. 420-431).

[18] Ahn, J., Hussain, R., Kang, K., & Son, J. (2025). Exploring encryption algorithms and network protocols: A comprehensive survey of threats and vulnerabilities. *IEEE Communications Surveys & Tutorials*, *27*(6), 3587-3614.

[19] Sun, H., Du, Y., & Li, Q. (2023). Deep learning-based detection technology for SQL injection research and implementation. *Applied Sciences*, *13*(16), 9466.

[20] Szmurlo, H., & Akhtar, Z. (2024). Digital sentinels and antagonists: The dual nature of chatbots in cybersecurity. *Information*, *15*(8), 443.

[21] Ahmed, D. (2025). *An Artificial Intelligence Model for the Exploitation of SQL Injection Vulnerabilities in Web Applications* (Doctoral dissertation, The George Washington University).