

Automated, Threat-Model-Driven Security Testing of Modern Systems: A Systematic Review Across Web, Microservices, Kubernetes-Orchestrated Cloud-Native, and Serverless Deployments

Musbahu Yunusa Makama^{1*}, Bashir Shehu Galadanchi², Aisha Saidu Muhammad¹, Sharfudeen Surajo Chiroma³, Abdulkadir Ahmad⁴, Ibrahim Umar⁵

¹ Department of Computer Science, Hussaini Adamu Federal Polytechnic, Kazaure, Nigeria.

² Faculty of Computing, Bayero University Kano, Kano, Nigeria.

³ Polytechnic Dutse, Jigawa State, Nigeria.

⁴ Department of Computer Science, Aliko Dangote University of Science and Technology, Wudil, Kano State, Nigeria.

⁵ Department of Computer Science, Federal University Dutse, Jigawa State, Nigeria.

ABSTRACT

This paper reports a systematic literature review, following established guidelines, of automated and threat-model-driven security testing research relevant to web, microservices, Kubernetes-orchestrated cloud-native, and serverless applications, published primarily between 2021 and 2026. Candidate studies were retrieved primarily through Google Scholar, a prominent academic search engine indexing scholarly literature across publishers, including IEEE, ACM, Elsevier, Springer, Wiley, and MDPI, supplemented by targeted web searches undertaken to verify publication venues and resolve citation details for papers carried forward into the final corpus. Eleven thematic query groups were employed, each aligned with the framework components and evaluation domains set out in the accompanying research proposal: automated security testing for microservices; threat-model-driven test generation; Kubernetes and continuous integration/continuous delivery (CI/CD) security; systematic reviews of microservices testing; application programming interface (API) security testing; development, security, and operations (DevSecOps) integration frameworks; container security scanning; serverless security; STRIDE and automated threat modeling; automated penetration testing; and web application security testing tools and benchmarking. Searches targeted publications from approximately 2021 to 2026, with a limited number of earlier foundational papers retained as background context. An initial pool of candidate studies was screened in two stages: relevance screening against explicit inclusion/exclusion criteria, followed by a venue-quality screening pass that identified and removed sources published in venues exhibiting predatory-publishing characteristics. The final corpus comprises 102 quality-screened primary studies across 11 thematic clusters. Results indicate that no single existing tool or technique achieves full vulnerability-detection coverage within any one deployment paradigm; that threat-model-to-executable-test pipelines are validated within single domains (notably automotive and, closest to this review's target problem, containerized microservices) but not generalized across paradigms; and that large language model (LLM)-assisted test and threat-model generation is a fast-growing but still narrow research direction. These findings motivate and directly inform a proposed threat-model-driven framework, described in an accompanying research proposal, built around a novel context-adaptor layer intended to generalize such test generation across web, microservices, Kubernetes, and serverless deployments, thereby establishing a foundation for rigorous security assurance across heterogeneous distributed architectures.

Keywords: integration/continuous delivery (CI/CD), venue-quality screening, accompanying research proposal, containerized microservices.

Relevant conflicts of interest/financial disclosures: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

INTRODUCTION

1.0 Introduction

The evolution of software construction and deployment appears to have outpaced the evolution of security testing. A single modern application may span several architectural paradigms at once: a traditional web front-end, a set of microservices communicating over internal APIs, containers orchestrated by Kubernetes, and event-driven serverless functions handling specific workloads. (Pourmajidi et al., 2025) Each paradigm may introduce its own attack surface — injection and session-handling flaws in web front-ends, broken object-level authorization in APIs, mis-configured RBAC and network policy in Kubernetes, and fundamentally different trust boundaries in Function-as-a-Service (FaaS) — yet the academic and practitioner literature appears largely to treat security testing for each paradigm as a separate problem, with separate tools, separate benchmarks, and separate research communities. (Marin et al., 2022), (Minna & Massacci, 2023)

Threat-model-driven test generation — deriving executable, targeted security tests directly from a structured threat model rather than generic rule sets — offers a principled alternative to this fragmentation, and has already been demonstrated successfully within individual domains. (Marback et al., 2012), (Casola et al., 2023) What remains unclear is whether, and to what extent, this approach generalizes across deployment paradigms. This systematic literature review was undertaken to delineate the current state of the art across the domains relevant to this question, and to identify the specific, evidenced gap that motivates the accompanying research proposal.

1.1 Background

1.1.1 Modern application paradigms and their distinct attack surfaces

Web applications remain the most extensively studied security-testing target, with a mature ecosystem of open-source and commercial static application security testing (SAST) and dynamic application security testing (DAST) tools (Aydos et al., 2021). Nevertheless, comparative benchmarking studies suggest persistent coverage gaps: across 75 real-world applications tested with nine open-source SAST and DAST tools, no single tool achieved full coverage against the OWASP Top-10/CWE-25 vulnerability

set. (Qadir et al., 2025) In a large Java-based system, multiple vulnerability-detection techniques identified vulnerabilities that the other evaluated techniques did not detect, and vice versa (Elder et al., 2022)—evidence suggesting that, even within a single paradigm, security-testing techniques may be complementary rather than redundant and that no single technique may suffice.

Microservices architectures decompose this problem further: security concerns may extend beyond individual services to encompass the trust relationships between them (Haindl et al., 2024). (Ünver & Britto, 2023) aggregated multiple open-source scanners into a unified feedback tool for microservices, while (Di et al., 2024) demonstrated industrial-scale distributed fuzzing for microservices at Ant Group, uncovering 1,764 confirmed security risks evidence both of the domain's practical importance and of the continued reliance on fuzzing and scanning approaches.

Kubernetes-orchestrated deployments may introduce a further, largely orthogonal layer of attack surface: container image vulnerabilities, RBAC misconfiguration, network-policy gaps, and admission-control weaknesses (Dell'Immagine et al., 2023). Empirical evaluations of container-scanning tools reveal persistent accuracy limitations (Javed & Toor, 2021) found that existing tools did not detect application package vulnerabilities and a Systematization of Knowledge at IEEE S&P re-evaluating existing container defenses against 51 real vulnerabilities found them of limited effectiveness (Haq et al., 2024).

Serverless (Function-as-a-Service) deployments appear to exhibit persistent security-testing gaps (Escaleira et al., 2025). (Escaleira et al., 2025) proposed a five-layer serverless security model in a systematic review, identifying function-level observability and data-lifecycle protection as persistent gaps, while (Hamza et al., 2025) empirically analyzed 88 real security issues drawn from GitHub across 10 open-source serverless platforms, finding certificate/encryption and permissions/role management the most frequently reported deficiencies.

1.1.2 Threat modeling as a candidate unifying mechanism

Threat modeling the structured identification of a system's potential attack vectors, most commonly via STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of

Privilege) or attack trees has been proposed as a mechanism for systematizing security testing, rather than leaving its organization to ad hoc tool selection. Awodele et al. (2024) conducted a comparative fuzzy-set risk evaluation across STRIDE, PASTA, LINDDUN, and other methodologies specifically for cloud environments, appearing to rank STRIDE as the most effective among those compared (Awodele et al., 2024), providing what may constitute empirical support for STRIDE's continued prominence in the reviewed literature. Recent work appears to automate the application of STRIDE itself: (Bandara et al., 2025) generates STRIDE analyses directly from architecture diagrams using vision-language models, while (Fu et al., 2025) automates STRIDE threat classification with 92.5% precision and rule-based construction on a large vulnerability dataset, suggesting the threat-identification stage long assumed to require manual expertise may be becoming increasingly amenable to automation (Bandara et al., 2025).

Threat-model-to-executable-test pipelines appear to have been demonstrated successfully within single domains. In automotive cybersecurity, (Marksteiner et al., 2023) carry a Threat Analysis and Risk Assessment (TARA) process through attack-tree decomposition to executable test cases. (Tanmay et al., 2025) extend this with a four-stage large language model (LLM)-driven pipeline attack-tree analysis, retrieval-augmented generation of domain knowledge, LLM-based test/property generation, and LLM-as-a-judge refinement validated on an automotive case study. Closest to this review's target problem, (Casola et al., 2023) present a model-based methodology carrying threat identification through to targeted test generation, validated on a containerized microservice case study the work that appears most directly relevant to this review, although it remains confined to one deployment paradigm.

Kubernetes-specific automated threat modeling appears to constitute an active but narrow direction: (Wimbauer et al., 2025) combine LLMs with attack graphs aligned to MITRE ATT&CK and the Microsoft Threat Matrix for Kubernetes to automate threat-hypothesis generation and quantitative risk scoring, but stop at risk quantification without generating executable tests a pattern repeated across nearly every reviewed study that reaches the cloud-native threat-modeling stage.

1.1.3 DevSecOps and CI/CD integration as the deployment context

A large and growing body of DevSecOps literature addresses embedding security activities throughout the CI/CD pipeline. (Sinan et al., 2025) and (Ramos & Yoo, 2025) each conducted systematic reviews identifying dozens of adoption challenges and corresponding solutions; (Rajapakse et al., 2021), despite preprint status, is highly cited (166 citations) for its thematic analysis of DevSecOps adoption barriers. (Cyril & Kumara, 2026) demonstrated a concrete, reproducible multi-tool CI/CD pipeline combining static application security testing (SAST), software composition analysis (SCA), dynamic application security testing (DAST), and machine-learning-based vulnerability classification with quantified precision and recall. These studies may collectively suggest that DevSecOps integration is well studied at the process and tooling-orchestration levels; however, consistent with the pattern identified in Section 1.1.2, the literature rarely appears to connect tooling decisions to an explicit upstream threat model.

1.2 Problem Statement

Despite substantial individual progress in automated security testing, three specific problems appear to remain unresolved in the literature reviewed for this study:

1. Tool fragmentation without full coverage published empirical comparisons consistently suggest that no single existing tool, or any combination of tools, appears to achieve comprehensive vulnerability coverage within a single deployment paradigm, let alone across multiple paradigms (Qadir et al., 2025; Seth et al., 2023; Haq et al., 2024).
2. Threat modeling is domain-siloed threat-model-to-test pipelines exist for automotive and hardware-verification domains, and threat-model-to-risk-quantification pipelines exist for Kubernetes specifically, but no reviewed framework, to the best of our knowledge, appears to carry a threat model through to executable tests across more than a single deployment paradigm (Casola et al., 2023; Marksteiner et al., 2023; Wimbauer et al., 2025).
3. No standardized cross-paradigm comparison exists the literature offers no empirical study measuring how a single security-testing

methodology's effectiveness varies as the same underlying application logic moves from a web front-end, to microservices, to Kubernetes orchestration, to serverless execution (Jayalath et al., 2024).

1.3 Aim and Objectives of the Review

Aim: To systematically review and synthesize the literature on automated and threat-model-driven security testing relevant to web, microservices, Kubernetes-orchestrated cloud-native, and serverless applications, in order to characterize the current state of the art and identify well-evidenced research gaps capable of grounding a PhD-level framework proposal.

Objectives

- To identify and classify automated security-testing approaches applicable to each of the four target deployment paradigms.
- To determine which threat-modeling formalisms are used to drive automated test generation, and the extent of such automation.
- To catalog the test types, tools, and test-oracle mechanisms reported across the reviewed literature, with particular attention to the oracle-automation problem.
- To examine how automated and threat-model-driven security-testing approaches are evaluated, including benchmarking practices and reported effectiveness metrics.
- To synthesize reported limitations and challenges, and to derive a specific, evidence-based research gap and roadmap for future work.

1.4 Research Questions

- RQ1: What automated security-testing approaches exist for web, microservices, Kubernetes-orchestrated cloud-native, and serverless applications, and how do they differ by paradigm?
- RQ2: Which threat-modeling formalisms (e.g., the STRIDE framework (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege), attack trees, and hybrid models) are used to drive automated test generation, and to what extent is the threat-modeling stage itself automated?

- RQ3: What types of security tests are generated or guided by automated or threat-model-driven approaches, and how is the test-oracle problem addressed?
- RQ4: How are automated and threat-model-driven security-testing approaches evaluated, and what evaluation practices (benchmarks, case studies, industrial deployment) are reported?
- RQ5: What limitations and challenges are reported, particularly regarding cross-paradigm generalization, continuous integration/continuous delivery (CI/CD) integration, and scalability?

2.0 Methodology

This review is structured in accordance with the systematic literature review guidelines for software engineering established by Kitchenham and Charters (2007), comprising planning, conducting, and reporting phases.

2.1 Search Strategy

Candidate studies were retrieved primarily through Google Scholar, a prominent academic search engine indexing scholarly literature across publishers, including IEEE, ACM, Elsevier, Springer, Wiley, and MDPI, supplemented by targeted web searches undertaken to verify publication venues and resolve citation details for papers carried forward into the final corpus. Eleven thematic query groups were employed, each aligned with the framework components and evaluation domains set out in the accompanying research proposal: automated security testing for microservices; threat-model-driven test generation; Kubernetes and continuous integration/continuous delivery (CI/CD) security; systematic reviews of microservices testing; application programming interface (API) security testing; development, security, and operations (DevSecOps) integration frameworks; container security scanning; serverless security; STRIDE and automated threat modeling; automated penetration testing; and web application security testing tools and benchmarking. Searches targeted publications from approximately 2021 to 2026, with a limited number of earlier foundational papers retained as background context.

2.2 Inclusion and Exclusion Criteria

Inclusion criteria:

- Peer-reviewed journal articles, conference papers, or high-quality preprints (e.g., arXiv, IACR ePrint) relevant to automated security testing, threat modeling, or DevSecOps within web, microservices, Kubernetes/container, or serverless contexts.
- Studies reporting a concrete approach, tool, framework, or empirical evaluation (case study, benchmark, or industrial deployment), excluding purely conceptual discussions.
- Systematic literature reviews and mapping studies directly relevant to the eleven query groups, retained as evidence for contextualization and gap identification.

Exclusion criteria:

- Non-scholarly publications (blog posts, vendor marketing pages, crowdsourced encyclopedias).
- Publications in venues exhibiting characteristics operationalized as predatory: fabricated or self-reported impact-factor claims not corresponding to a recognized indexing service (Beall, 2016); guaranteed rapid publication (e.g., “publication within 2 days”) as a marketed feature (Shen & Björk, 2015); or a stated journal scope systematically mismatched with the paper's subject matter (e.g., a crisis-communication or mathematics journal publishing DevSecOps engineering papers).
- Non-English language publications.

2.3 Study Selection and Quality Screening

Study selection proceeded in two stages: first, a relevance screen removed candidates outside the scope of the 11 query groups or the 4 target deployment paradigms. Second, a venue-quality screen was applied to every remaining candidate: each publication venue was checked against its claimed indexing status, and papers appearing in venues matching the predatory-pattern criteria in Section 2.2 were removed. This screening pass may constitute a substantive contribution to this review: an initial candidate pool of 123 papers was reduced to a validated corpus of 102 after 31 papers from predatory-pattern or unverifiable venues were identified and either removed outright (21 papers, where no adequate substitute existed for the specific claim) or replaced with a verified, legitimate paper making the same evidentiary claim (9 papers, including a fuzzer published at the International Conference on Software Engineering (ICSE), one of the field's top venues, replacing a paper originally sourced from a venue exhibiting fabricated-impact-factor characteristics). One further correction reclassified a legitimate IEEE-sponsored conference (previously mis-screened as low-tier) back into the primary corpus. The full identification, screening, eligibility, and inclusion process is summarized in the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) 2020 flow diagram in Figure 1.

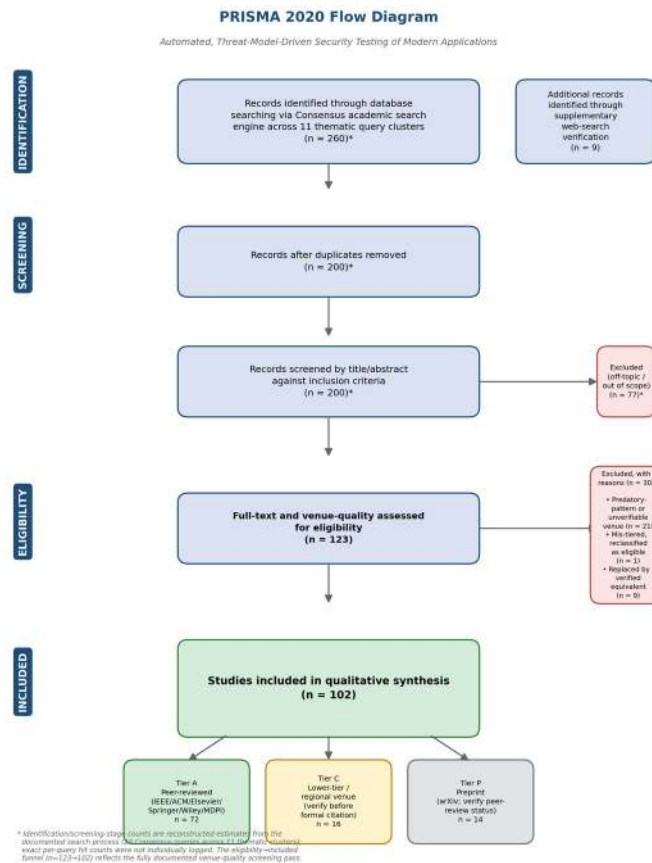


Figure 1: PRISMA 2020 Flow Diagram for This Systematic Literature Review.

Quality Tier	Definition	Papers
A — Reputable	IEEE / ACM / Elsevier / Springer / Wiley / MDPI / IOS Press / established conferences and archives	72
C — Low-tier / regional	Legitimate venues of lower visibility, regional scope, or unresolved classification; used solely for background context	16
P — Preprint	arXiv or similar, not peer-reviewed; retained for methodological relevance pending verification of peer-reviewed status	14
TOTAL		102

Note on Figure 1: The Eligibility → Included stage of the diagram ($n = 123 \rightarrow n = 102$) appears to reflect the fully documented counts from the venue-quality screening pass described above. The Identification and Screening stages are reconstructed estimates based on the 23 consensus search queries executed across the 11 thematic clusters; exact per-query result counts were not individually logged during the search process and are marked with an asterisk in the figure accordingly.

2.4 Data Extraction and Synthesis

A structured data-extraction scheme captured, for each included study: bibliographic data (authors, year, venue); target deployment paradigm (s); whether the study proposes a framework or tool; the security-

testing method (s) used; the study's key contribution; and its principal reported limitation or the research gap it leaves open. Data were synthesized narratively and organized by research question (Section 3), with a summary table (Section 3.6) mapping each of the 11 thematic clusters to its role in answering the review's research questions.

3.0 Results

3.1 RQ1 Automated Security-Testing Approaches by Paradigm

Across the four target paradigms, automated security-testing research appears to cluster around distinct tool families rather than a shared methodology. For web applications, comparative benchmarking is the dominant research mode: (Qadir et al., 2025) and

(Każmierak, 2025) each benchmark open-source SAST/DAST tool combinations against real or seeded vulnerabilities, with findings that suggest incomplete coverage; (Albahar et al., 2022) and (Shahid et al., 2022) provide widely cited (35 and 57 citations, respectively) empirical comparisons of penetration-testing tools and scanners. (Pan et al., 2024), published at ICSE, introduced EDEFuzz, which appears to be the first automated fuzzer specifically targeting Excessive Data Exposure (OWASP API #3), using a metamorphic relation approach to overcome that vulnerability class's lack of an explicit test oracle directly relevant to RQ3 (Section 3.3).

For microservices, fuzzing and multi-scanner aggregation appear to predominate: (Di et al., 2024) demonstrates industrial-scale distributed fuzzing (MICRoFuzz) at Ant Group, while (Ünver & Britto, 2023) aggregates open-source scanners into unified tooling. (Ai et al., 2026) applies LLM-driven semantic inference to discover business-logic vulnerabilities in microservice APIs specifically among the more substantial automation results in the corpus, although its applicability appears domain-specific to power and smart-grid systems.

For Kubernetes and container environments, automated approaches appear to concentrate on policy enforcement and admission control: Gurajapu et al. (2026) achieve sub-200ms admission-webhook validation combining SBOM generation, signature verification, and policy-as-code (Gurajapu et al., 2026). Container image scanning itself remains imperfect: (Javed & Toor, 2021) and (Haq et al., 2024) both document substantial detection gaps in widely used scanners.

For serverless, automated approaches appear less mature and tend to concentrate on post-deployment monitoring rather than pre-deployment testing: (Ben-Shimol et al., 2024) detects compromised functions via native-monitoring-based anomaly detection, and (Shin et al., 2025) verifies function calls in real time to defend against privilege-abuse attacks both constitute runtime defenses rather than instances of pre-deployment test generation.

3.2 RQ2 — Threat-Modeling Formalisms Used to Drive Test Generation

STRIDE appears to remain the most extensively validated and applied threat-modeling formalism in the reviewed literature, consistent with its empirically

demonstrated strength for cloud environments (Awodele et al., 2024). Multiple studies extend STRIDE with automation: (Bandara et al., 2025) generate STRIDE analyses from architecture diagrams using vision-language and reasoning models; (Fu et al., 2025) automate STRIDE threat classification with 92.5% precision; and (Rouland et al., 2021) provide formal specification and tool support for STRIDE threat detection and treatment at the software-component level.

Attack trees, often combined with STRIDE in hybrid frameworks, appear to be the second-most prevalent formalism. (Abuabed et al., 2023), which applied STRIDE, attack-tree analysis, and CVSS scoring to automotive Advanced Driver-Assistance Systems, identified numerous potential threats illustrating the formalism's capacity for systematic, quantified threat enumeration at scale. (Tanmay et al., 2025) appear to advance furthest along the threat-model-to-test pipeline, using LLM-driven, self-corrective retrieval-augmented generation to transform attack trees directly into executable test cases and temporal-logic properties though validated only in the automotive domain.

Automation of the threat-modeling stage itself, independent of downstream test generation, appears to be a fast-growing subfield: in addition to Deep-STRIDE and (Fu et al., 2025), (Wimbauer et al., 2025) apply LLMs and MITRE ATT&CK-aligned attack graphs specifically to Kubernetes deployments. None of these Kubernetes- or cloud-native-focused automated threat-modeling studies, however, appear to extend their pipeline to executable test generation the specific gap this review's RQ2 identifies as most acute for the cloud-native and serverless paradigms.

3.3 RQ3 Test Types, Tools, and the Oracle Problem

Reviewed studies support the generation or orchestration of several test types: fuzzing and penetration tests (Di et al., 2024; Jaswal et al., 2026, “AWE” 87% XSS success and 66.7% blind SQLi success on the XBOW benchmark using memory-augmented multi-agent LLMs); static and dynamic analysis integrated into pipelines (Cyril et al., 2026; Pavlyshyn et al., 2026, reporting F1 0.91–0.95 for a hybrid rule-based/LLM SAST triage layer); configuration and infrastructure tests for containers and Kubernetes (Kim et al., 2025; Chen et al., 2022); and API-specific

tests targeting OWASP API Top-10 categories (Yıldırım et al., 2024; Ishida et al., 2024).

The test-oracle problem determining whether a given test output constitutes a genuine security violation appears to remain only partially automated across the corpus (Chen et al., 2016). Most frameworks appear to automate test-input generation but rely on predefined signatures, status-code checks, or human review to judge outcomes for complex, business-logic-level vulnerabilities. The clearest exception is metamorphic testing (Chaleshtari et al., 2023): This technique is consequently treated as a design precedent in the accompanying research proposal's Test Oracle module.

3.4 RQ4 Evaluation Practices and Benchmarking

Evaluation practices appear to vary considerably in rigor across the corpus. More rigorous, benchmark-first designs are comparatively rare but growing. (Gioacchini et al., 2024) appear to establish an open, standardized 33-task benchmark for LLM penetration-testing agents, reporting a stark gap between fully autonomous (21% success rate) and human-assisted (64% success rate) agent architectures a caution that may be directly relevant to the degree of autonomy that any future threat-model-driven test generator should be granted.

Reported effectiveness metrics appear inconsistent across studies, complicating direct comparison: some report precision, recall, and F1, others report task-completion or success rates (Gioacchini et al., 2024). No reviewed study appears to evaluate a single security-testing approach across more than one deployment paradigm under a consistent methodology — a gap that the accompanying proposal's cross-domain comparison may be designed to address.

3.5 RQ5 Reported Limitations and Open Challenges

Across the corpus, five recurring limitations can be discerned:

- Incomplete coverage within a single paradigm — no individual tool or tool combination appears to achieve comprehensive vulnerability coverage, even within a single deployment context (Qadir et al., 2025; Seth et al., 2023).
- Manual oracle definition for complex vulnerabilities — automation of test-input generation appears to outpace that of pass/fail judgment, particularly for business-logic and multi-step attack scenarios (Chaleshtari et al., 2023).
- Domain-locked threat-model-to-test pipelines — the most advanced pipelines (including STAF and Casola et al.'s microservice methodology) appear to have been validated primarily within single domains.
- LLM-assisted components may introduce non-determinism; reproducibility is rarely reported explicitly, a limitation that AWE (Jaswal et al., 2026) addresses directly, although such reporting remains uncommon elsewhere in the corpus.
- Kubernetes- and serverless-specific threat modeling stops short of test generation — Threat-Compute and related work automate threat identification and risk scoring for cloud-native infrastructure but appear not to close the loop to executable testing.

3.6 Summary of Reviewed Literature by Thematic Cluster

Thematic Cluster	Papers	Primary Research Questions (RQs) Addressed
Microservices Security Testing	8	RQ1
Threat-Model-Driven Test Generation	11	RQ2, RQ3
Kubernetes and Continuous Integration/Continuous Delivery (CI/CD) Security	6	RQ1, RQ4
Systematic Literature Reviews (SLRs) on Microservices Testing	9	Background, RQ5
Application Programming Interface (API) Security Testing (Open Worldwide Application Security Project [OWASP])	6	RQ1, RQ3
Development, Security, and Operations (DevSecOps) Integration Frameworks	17	RQ1, RQ4
Container Security Scanning	10	RQ1, RQ5
Serverless Security	13	RQ1, RQ5

Thematic Cluster	Papers	Primary Research Questions (RQs) Addressed
STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) and Automated Threat Modeling	9	RQ2
Automated Penetration Testing	6	RQ1, RQ4
Web App Security Testing (Tools/Benchmarking)	7	RQ1, RQ3, RQ4

4.0 Discussion

Taken together, the answers to RQ1–RQ5 converge on a coherent, well-substantiated pattern. Automated security testing appears effective, yet remains persistently incomplete within any single deployment paradigm (RQ1, RQ4); STRIDE and attack-tree threat modeling are mature and increasingly self-automating at the identification stage (RQ2), but the transition from threat model to executable test appears to remain confined to individual domains wherever it has been demonstrated; test-oracle automation lags test-generation automation almost everywhere except in the narrow but promising case of metamorphic testing (RQ3); and no reviewed evaluation methodology spans more than one deployment paradigm (RQ4, RQ5).

This pattern directly motivates the framework proposed in the accompanying research proposal: a single threat-model-driven pipeline, built around STRIDE and a metamorphic-relation-capable test oracle (following the precedent of Pan et al., 2023), generalized across paradigms via a dedicated context-adaptor layer rather than reimplemented separately for each. No study identified in this review appears to attempt this specific generalization, which may position it as a substantiated contribution rather than an incremental extension of any single prior work.

5.0 Threats to Validity

Construct validity may be affected by classification judgments (e.g., assigning a given study to “threat-model-driven” versus “general automated testing”); borderline cases were resolved by close reading of each paper’s stated methodology rather than title or abstract alone.

Internal validity risks arise from reliance on a single primary search tool (Consensus), which, while indexing a broad base of peer-reviewed literature, may not achieve the same completeness as a multi-database systematic search across IEEE Xplore, ACM

Digital Library, and Scopus directly. This was partially mitigated by targeted supplementary web searches used to verify and, where necessary, correct citation details for every paper carried into the final corpus, but full database-level completeness cannot be claimed.

External validity is limited by the corpus’s heavy weighting toward very recent publications (2024–2026), reflecting the fast-moving, LLM-influenced state of this research area; older foundational work may be under-represented outside the specific papers retained for background context.

Reliability may be strengthened by the explicit, reproducible venue-quality screening process documented in Section 2.3, which appears unusually transparent — 31 initially retrieved papers were identified as venue-quality risks and either replaced or removed, with the reasoning documented rather than silently applied. Independent replication of the search and screening process would nonetheless further strengthen confidence in the corpus’s completeness.

CONCLUSION

This systematic literature review synthesized 102 quality-screened primary studies spanning automated security testing for web, microservices, Kubernetes-orchestrated cloud-native, and serverless applications. The evidence suggests that no single existing tool or technique achieves full vulnerability coverage within any one deployment paradigm; that threat-model-driven test generation is validated and effective within individual domains but has not been generalized across paradigms (Casola et al., 2023); and that test-oracle automation, while advancing via metamorphic testing techniques, remains the most under-automated stage of the pipeline for complex vulnerability classes (Harman & O’Hearn, 2018). These findings appear to delineate a specific research gap — the absence of a single framework generalizing threat-model-driven test generation across modern application deployment

paradigms — that may directly motivate the framework proposed in the accompanying PhD research proposal.

Future Work

Building directly on this review's findings, future work may prioritize: (1) design and implement a threat-model-driven pipeline with an explicit context-adaptor mechanism capable of generalizing across web, microservices, Kubernetes, and serverless deployments, leveraging the demonstrated feasibility of automated STRIDE rule construction (Fu et al., 2025) and attack-tree-to-test generation (Khule et al., 2025); (2) extend metamorphic test-oracle techniques (Pan et al., 2023) beyond their current single-vulnerability-class application to a broader range of business-logic and cross-service vulnerabilities; (3) close the loop between Kubernetes-native automated threat modeling (Wimbauer et al., 2025) and executable test generation, an integration that, to the best of current evidence, no reviewed study appears to perform; and (4) establish a standardized, reusable cross-paradigm benchmark, following the rigorous benchmark-design precedent set by Gioacchini et al. (2024) for penetration-testing agents, to enable direct comparison of security-testing effectiveness across deployment paradigms — a comparison that this review suggests is absent from the current literature.

REFERENCES

1. Abuabed, Z., Alsadeh, A., & Taweel, A. (2023). STRIDE threat model-based framework for assessing the vulnerabilities of modern vehicles. *Computers & Security*, 133, 103391–103391. <https://doi.org/10.1016/j.cose.2023.103391>
2. Ai, X., Yin, Y., Huang, Y., Meng, Q., Dong, Y., & Chen, Z. (2026). A large language model-driven semantic inference framework for business logic vulnerability discovery in power system microservice APIs. *Journal of Engineering and Applied Science*, 73(1). <https://doi.org/10.1186/s44147-026-01100-9>
3. Albahar, M. A., Alansari, D., & Jurecut, A. D. (2022). An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities. *Electronics*, 11(19), 2991–2991. <https://doi.org/10.3390/electronics11192991>
4. Awodele, O., Ogbonna, C., Ogu, E. O., Hinmikaiye, J. O., & Akinsola, J. E. T. (2024). Characterization and Risk Assessment of Cyber Security Threats in Cloud Computing: A Comparative Evaluation of Mitigation Techniques. *Acadlore Transactions on AI and Machine Learning*, 3(2), 106–118. <https://doi.org/10.56578/ataiml030204>
5. Aydos, M., Aldan, Ç., Coşkun, E., & Soydan, A. (2021). Security testing of web applications: A systematic mapping of the literature. *Journal of King Saud University - Computer and Information Sciences*, 34(9), 6775–6792. <https://doi.org/10.1016/j.jksuci.2021.09.018>
6. Bandara, E., Hass, A., Shetty, S., Mukkamala, R., Gore, R., Rahman, A., & Bouk, S. H. (2025). *Deep-Stride: Automated Security Threat Modeling with Vision-Language Models*. 1–7. <https://doi.org/10.23919/softcom66362.2025.11197424>
7. Beall, J. (2016). Best practices for scholarly authors in the age of predatory journals. *Annals of The Royal College of Surgeons of England*, 98(2), 77–79. <https://doi.org/10.1308/rcsann.2016.0056>
8. Ben-Shimol, L., Lavi, D., Klevansky, E., Brodt, O., Mimran, D., Elovici, Y., & Shabtai, A. (2024). Detection of compromised functions in a serverless cloud environment. *Computers & Security*, 150, 104261–104261. <https://doi.org/10.1016/j.cose.2024.104261>
9. Caceres, R. J. R. (2025). Mecanismos De Seguridad En Arquitecturas De Microservicios: Revisión Sistemática De Literatura. *Revista de investigación de Sistemas e Informática*, 18(2), 143–157. <https://doi.org/10.15381/risi.v18i2.31160>
10. Casola, V., Benedictis, A. D., Mazzocca, C., & Orbinato, V. (2023). Secure software development and testing: A model-based methodology. *Archivio Istituzionale Della Ricerca (Alma Mater Studiorum Università Di Bologna)*, 137, 103639–103639. <https://doi.org/10.1016/j.cose.2023.103639>
11. Chaleshtari, N. B., Pastore, F., Göknıl, A., & Briand, L. (2023). Metamorphic Testing for Web System Security. *Open Repository and Bibliography (University of Luxembourg)*, 1–43. <https://doi.org/10.1109/tse.2023.3256322>

12. Chen, L., Xia, Y., Ma, Z., Zhao, R., Wang, Y., Liu, Y., Sun, W., & Xue, Z. (2022). SEAF: A Scalable, Efficient, and Application-independent Framework for container security detection. *Journal of Information Security and Applications*, 71, 103351–103351. <https://doi.org/10.1016/j.jisa.2022.103351>
13. Chen, T. Y., Kuo, F., Ma, W., Susilo, W., Towey, D., Voas, J., & Zhou, Z. Q. (2016). Metamorphic Testing for Cybersecurity. *Computer*, 49(6), 48–55. <https://doi.org/10.1109/mc.2016.176>
14. Cyril, H. P., & Kumara, S. (2026). *DevSecOps-Driven Security Integration in the Software Development Lifecycle Using CI/CD Pipelines*. 1–6. <https://doi.org/10.1109/icaic67076.2026.11395737>
15. Dell’Immagine, G., Soldani, J., & Brogi, A. (2023). KubeHound: Detecting Microservices’ Security Smells in Kubernetes Deployments. *Future Internet*, 15(7), 228–228. <https://doi.org/10.3390/fi15070228>
16. Developer, S., & Avuthu, Y. R. (2023). Microservices Security Threat Modelling in DevOps Pipelines. *Journal of Mathematical & Computer Applications*, 1–7. [https://doi.org/10.47363/jmca/2023\(2\)e138](https://doi.org/10.47363/jmca/2023(2)e138)
17. Di, P., Liu, B., & Gao, Y. (2024). *MicroFuzz: An Efficient Fuzzing Framework for Microservices*. 216–227. <https://doi.org/10.1145/3639477.3639723>
18. Elder, S., Zahan, N., Shu, R., Metro, M., Kozarev, V., Menzies, T., & Williams, L. (2022). Do I really need all this work to find vulnerabilities? An empirical case study comparing vulnerability detection techniques on a Java application. In *arXiv (Cornell University)*. Cornell University. <https://doi.org/10.48550/arxiv.2208.01595>
19. Escalreira, P., Cunha, V. A., Barraca, J. P., Gomes, D., & Aguiar, R. L. (2025). A systematic review on security mechanisms for serverless computing. *Cluster Computing*, 28(7). <https://doi.org/10.1007/s10586-025-05371-4>
20. Fu, C., Zhang, H., & Guan, X. (2025). An Automated Approach to Constructing STRIDE Threat Rule Model and Updating Rule Base. *ACM Transactions on Software Engineering and Methodology*, 35(7), 1–36. <https://doi.org/10.1145/3772283>
21. Gioacchini, L., Mellia, M., Drago, I., Delsanto, A., Siracusano, G., & Bifulco, R. (2024). AutoPenBench: Benchmarking Generative Agents for Penetration Testing. In *arXiv (Cornell University)*. Cornell University. <https://doi.org/10.48550/arxiv.2410.03225>
22. Gurajapu, A., Anumolu, S., Agarwal, A., & Yeka, V. (2026). Shift-Left Security Validation of Containers via Kubernetes Admission Webhook. *Frontiers in Computer Science and Artificial Intelligence*, 5(2), 63–68. <https://doi.org/10.32996/jcsts.2026.5.1.6>
23. Haindl, P., Kochberger, P., & Sveggen, M. (2024). A Systematic Literature Review of Inter-Service Security Threats and Mitigation Strategies in Microservice Architectures. *IEEE Access*, 12, 90252–90286. <https://doi.org/10.1109/access.2024.3406500>
24. Hamza, M., Akbar, M. A., Smolander, K., & Khan, A. A. (2025). *Security Perspective of Open-Source Serverless Platforms: An Empirical Investigation*. 152–161. <https://doi.org/10.1145/3727967.3756839>
25. Hannousse, A., & Yahiouche, S. (2021). Securing microservices and microservice architectures: A systematic mapping study. *arXiv (Cornell University)*, 41, 100415–100415. <https://doi.org/10.1016/j.cosrev.2021.100415>
26. Haq, M. S., Nguyen, T. D., Tosun, A. Ş., Vollmer, F., Korkmaz, T., & Sadeghi, A. (2024). *SoK: A Comprehensive Analysis and Evaluation of Docker Container Attack and Defense Mechanisms*. 4573–4590. <https://doi.org/10.1109/sp54263.2024.00268>
27. Harman, M., & O’Hearn, P. W. (2018). *From Start-ups to Scale-ups: Opportunities and Open Problems for Static and Dynamic Program Analysis*. 1–23. <https://doi.org/10.1109/scam.2018.00009>
28. Hutasuht, N. R. P., Amri, M. G., & Aji, R. F. (2024). Security Gap in Microservices: A Systematic Literature Review. *International Journal of Advanced Computer Science and Applications*, 15(12). <https://doi.org/10.14569/ijacsa.2024.0151218>

29. Ishida, Y., Hanada, M., Waseda, A., & Kim, M. W. (2024). *Automated Vulnerability Assessment Approach for Web API that Considers Requests and Responses*. 1521–1533. <https://doi.org/10.23919/icact60172.2024.10471939>
30. Javed, O., & Toor, S. (2021). *An Evaluation of Container Security Vulnerability Detection Tools*. 95–101. <https://doi.org/10.1145/3481646.3481661>
31. Jayalath, R. K., Ahmad, H., Goel, D., Syed, M. S. B., & Ullah, F. (2024). Microservice Vulnerability Analysis: A Literature Review With Empirical Insights. *IEEE Access*, 12, 155168–155204. <https://doi.org/10.1109/access.2024.3481374>
32. Kaźmierak, I. (2025). Comparison of the effectiveness of tools for testing the security of web applications. *Journal of Computer Sciences Institute*, 34, 36–43. <https://doi.org/10.35784/jcsi.6613>
33. Kim, Y., Park, C.-H., & Hong, D. (2025). A Robust Framework for Comprehensive Container Image Vulnerability Assessment. *IEEE Access*, 13, 35837–35847. <https://doi.org/10.1109/access.2025.3545124>
34. Mahmood, S., Nguyen, H. N., & Shaikh, S. A. (2022). Systematic threat assessment and security testing of automotive over-the-air (OTA) updates. *Vehicular Communications*, 35, 100468–100468. <https://doi.org/10.1016/j.vehcom.2022.100468>
35. Marback, A., Do, H., He, K., Kondamarri, S., & Xu, D. (2012). A threat model-based approach to security testing. *Software Practice and Experience*, 43(2), 241–258. <https://doi.org/10.1002/spe.2111>
36. Marin, E., Perino, D., & Pietro, R. D. (2022). Serverless computing: a security perspective. *Journal of Cloud Computing Advances Systems and Applications*, 11(1). <https://doi.org/10.1186/s13677-022-00347-w>
37. Marksteiner, S., Schmittner, C., Christl, K., Ničković, D., Sjödin, M., & Sirjani, M. (2023). *From TARA to Test: Automated Automotive Cybersecurity Test Generation Out of Threat Modeling*. 1–10. <https://doi.org/10.1145/3631204.3631864>
38. Miao, T., Shaafi, A. I., & Song, E.-J. (2025). Systematic Mapping Study of Test Generation for Microservices: Approaches, Challenges, and Impact on System Quality. *Electronics*, 14(7), 1397–1397. <https://doi.org/10.3390/electronics14071397>
39. Minna, F., & Massacci, F. (2023). SoK: Runtime security for cloud microservices. Are we there yet? *Computers & Security*, 127, 103119–103119. <https://doi.org/10.1016/j.cose.2023.103119>
40. Pan, L., Cohnney, S., Murray, T., & Pham, V.-T. (2024). *EDEFuzz: A Web API Fuzzer for Excessive Data Exposures*. 1–12. <https://doi.org/10.1145/3597503.3608133>
41. Pavlyshyn, V., & Krasko, O. (2026). SAST IMPROVEMENTS USING LLM FOR CI/CD PIPELINES. *Information and Communication Technologies Electronic Engineering*, 6(1), 152–160. <https://doi.org/10.23939/ictee2026.01.152>
42. Pourmajidi, W., Zhang, L., Steinbacher, J., Erwin, T., & Miranskyy, A. (2025). A Reference Architecture for Governance of Cloud Native Applications. *IEEE Transactions on Cloud Computing*, 13(3), 935–952. <https://doi.org/10.1109/tcc.2025.3578557>
43. Qadir, S., Waheed, E., Khanum, A., & Jehan, S. (2025). Comparative evaluation of approaches & tools for effective security testing of Web applications. *PeerJ Computer Science*, 11. <https://doi.org/10.7717/peerj-cs.2821>
44. Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. *arXiv (Cornell University)*. <https://doi.org/10.48550/arxiv.2103.08266>
45. Ramos, R. C. B., & Yoo, S. G. (2025). Cybersecurity in DevOps Environments: A Systematic Literature Review. *IEEE Access*, 13, 191959–191979. <https://doi.org/10.1109/access.2025.3582892>
46. Rouland, Q., Hamid, B., & Jaskolka, J. (2021). Specification, detection, and treatment of STRIDE threats for software components: Modeling, formal methods, and tool support. *Journal of Systems Architecture*, 117, 102073–102073. <https://doi.org/10.1016/j.sysarc.2021.102073>

47. Seth, A., Bhattacharya, S., Elder, S., Zahan, N., & Williams, L. (2025). Comparing effectiveness and efficiency of Interactive Application Security Testing (IAST) and Runtime Application Self-Protection (RASP) tools in a large java-based system. *Empirical Software Engineering*, 30(3). <https://doi.org/10.1007/s10664-025-10621-5>
48. Shahid, J., Hameed, M. K., Javed, I. T., Qureshi, K. N., Ali, M., & Crespi, N. (2022). A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions. *Applied Sciences*, 12(8), 4077–4077. <https://doi.org/10.3390/app12084077>
49. Shen, C., & Björk, B. (2015). ‘Predatory’ open access: a longitudinal study of article volumes and market characteristics. *BMC Medicine*, 13(1), 230–230. <https://doi.org/10.1186/s12916-015-0469-2>
50. Shin, C., Kim, B., & Lee, S. (2025). Bambda: A Real-Time Verification Framework for Serverless Computing. *IEEE Access*, 13, 90896–90911. <https://doi.org/10.1109/access.2025.3572729>
51. 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). <https://doi.org/10.1002/smr.70029>
52. Tanmay, K., Stefan, M., Jose, A., Hannes, F., Sebastian, F., & Apurva, N. (2025). STAF: Leveraging LLMs for Automated Attack Tree-Based Security Test Generation. In *arXiv (Cornell University)*. Cornell University. <https://doi.org/10.48550/arxiv.2509.20190>
53. Tran, A.-D., Verreydt, S., Yskout, K., & Joosen, W. (2026). A Multivocal Literature Review on the Effectiveness of Security Threat Modeling. *IEEE Transactions on Software Engineering*, 1–17. <https://doi.org/10.1109/tse.2026.3662996>
54. Ünver, B., & Britto, R. (2023). Automatic Detection of Security Deficiencies and Refactoring Advises for Microservices. *KTH Publication Database DiVA (KTH Royal Institute of Technology)*, 25–34. <https://doi.org/10.1109/icssp59042.2023.00013>
55. Wimbauer, A., Muscariello, L., Samain, J., Stegger, L., Glas, K., Helm, M., & Carle, G. (2025). *ThreatCompute: Leveraging LLMs for Automated Threat Modeling of Cloud-Native Applications*. 14–27. <https://doi.org/10.1145/3733812.3765533>
56. Yıldırım, R., Aydın, K., & Çetin, O. (2024). *Evaluating the Impact of Conventional Code Analysis Against Large Language Models in API Vulnerability Detection*. 57–64. <https://doi.org/10.1145/3655693.3655701>

HOW TO CITE: Musbahu Yunusa Makama, Bashir Shehu Galadanchi, Aisha Saidu Muhammad, Sharfudeen Surajo Chiroma, Abdulkadir Ahmad, Ibrahim Umar, Automated, Threat-Model-Driven Security Testing of Modern Systems: A Systematic Review Across Web, Microservices, Kubernetes-Orchestrated Cloud-Native, and Serverless Deployments, *Int. J. Sci. R. Tech.*, 2026, 3 (10), 1028-1040. <https://doi.org/10.5281/zenodo.23473123>