

Automated and AI Assisted Web Security Auditing a Focused Narrative Review

Akshat Kumar¹, Gaurav Sharma², Deep Singh³, Roovi⁴

Dr. APJ Abdul Kalam Technical University, Lucknow

Abstract- Automated web security auditing combines several tasks that produce different kinds of evidence: discovering exposed functionality, exercising application behaviour, interpreting observations, and deciding which weaknesses deserve attention. This focused narrative review examines five selected studies on cross-origin resource sharing, stateful REST testing, GraphQL testing, language-model-assisted penetration testing, and vulnerability prioritization. Four protocol specifications provide supporting definitions. The literature is compared by input requirements, treatment of application state, evidence produced, and limits of interpretation. The synthesis indicates that wider endpoint coverage, successful testing actions, and confirmed security weaknesses are distinct outcomes. Schema-aware testing helps explore structured APIs, while language models can support interpretation and planning; neither removes the need to validate findings against observable behavior. The review identifies evaluation comparability, authentication context, evidence of traceability, and constrained AI assistance as priorities for an AI Security Auditor project. It presents design implications and future evaluation questions without reporting a new implementation, benchmark, or experiment. The source selection is purposive rather than exhaustive, and conclusions are limited to the reviewed material.

Keywords— Web security auditing; API testing; vulnerability assessment; large language models; narrative review.

I. INTRODUCTION

A useful security audit must explain what was observed, why it matters, and what evidence would justify corrective action. An exposed endpoint is not automatically vulnerable, and a warning from a scanner is not equivalent to a demonstrated attack. These distinctions matter when automated reports are intended for developers who must decide which findings to investigate.

The selected literature addresses different parts of this problem. Chen et al. examine browser trust and CORS deployment [1]. RESTler explores dependent REST requests [2]. GraphQLer studies state-aware GraphQL testing [3].

PentestGPT examines language models within penetration-testing workflows [4]. Bulut et al. investigate vulnerability prioritisation informed by offensive security [5]. Together, these works offer a focused basis for considering how an auditor should gather, interpret, and communicate security evidence.

The term AI-assisted is used here for workflows in which a model supports interpretation, planning, or prioritisation. It does not imply that all scanning components use machine appropriate even when a language model writes the explanation. Keeping these roles explicitly makes the resulting system easier to evaluate.

This review asks three questions. First, what information and application context do the selected approaches require? Second, what does their output establish about security? Third, which gaps should guide the design and evaluation of a student-scale AI Security Auditor? The contribution is a qualitative comparison and a set of bounded design implications, rather than a claim that a new auditor has already achieved improved detection.

II. REVIEW SCOPE AND APPROACH

This is a focused narrative review of five purposively selected studies [1]–[5], supported by four specifications [6]–[9]. Sources were checked through public publisher, author and standards pages on 14

September 2026. Targeted web queries included "RESTler stateful REST API fuzzing", "CORS empirical study" and "PentestGPT penetration testing", followed by direct retrieval of relevant primary records. The selection was informed by the web and API auditing scope of the project.

Studies were retained when they described a relevant method and made its intended task identifiable. Protocol sources were retained to clarify API structure and browser security semantics. Product marketing, unrelated security domains, and unverifiable numerical claims were excluded from the synthesis. No exhaustive database search, duplicate screening process, or formal risk-of-bias instrument was performed. Accordingly, this paper does not claim systematic review of coverage or meta-analysis. Extraction focused on four comparable attributes: required inputs, handling of context or state, the evidence returned, and the conditions limiting interpretation. Table I summarises the selected studies. Publication types are kept visible: GraphQLer is cited as its August 2026 arXiv version, and the prioritisation study as its 2022 arXiv version. Standards explain expected behaviour; they are not counted as experimental evaluations.

III. TECHNICAL FOUNDATIONS

OpenAPI 3.1.0 describes HTTP APIs through structured information such as paths, operations, parameters, responses, and security requirements [6]. Such descriptions can guide test construction. However, a declared operation is a statement about an interface, not independent evidence that its implementation correctly enforces access control. An auditor therefore needs to distinguish specifications of availability from implementation of assurance.

GraphQL defines typed operations and a response structure that can include data and errors [7]. Its schema offers a different view of the attack surface from a collection of HTTP paths. For review purposes, operation reachability and security validation must be considered separately: successfully executing a query establishes that a path through the interface

was exercised but does not establish that every relevant permission boundary was tested.

The origin model in RFC 6454 distinguishes origins using scheme, host, and port [8]. A hostname-only comparison can therefore miss a security boundary. The Fetch Standard also makes CORS interpretation dependent on request mode, credentials and response headers; a wildcard origin does not authorize browser sharing for a request whose credentials mode is included [9]. These definitions constrain what a scanner can infer from a header.

This review uses four analytical stages: discovery, validation, GraphQLer [3] 2026 version PentestGPT [4] 2024 Bulut et al. [5] 2022 Schema dependency graph and targeted operation chains LLM modules for testing workflow assistance Offensive-security-informed prioritization Coverage and validated cases on stated targets; metrics depend on the evaluation subset Task completion on benchmark targets; not a general detection-accuracy measure Case-study evidence for ranking; transfer to another environment needs evaluation interpretation, and prioritization. Discovery identifies candidate interfaces or configurations. Validation checks whether an observation satisfies a security-relevant condition. Interpretation explains that condition and its uncertainty. Prioritization relates the validated finding to remediation decisions. This grouping is a synthesis used to compare the sources, not a taxonomy attributed to any single study.

IV. COMPARISON OF SELECTED STUDIES

1. Browser policy and configuration evidence

Chen et al. [1] show that CORS security depends on design choices, implementation, and deployment. Their study identifies problematic trust dependencies and misunderstandings of the mechanism. These supports treating cross-origin configuration as a contextual security question rather than a search for one suspicious header value. The study concerns browser-mediated behavior and does not establish the performance of an entire web vulnerability scanner.

Table 1. Method and evidence across the selected literature

Study	Method and input	Evidence and boundary
Chen et al. [1] 2018	Empirical CORS analysis; browser and site behavior	Trust and configuration weaknesses; interpretation depends on browser semantics
RESTler [2] 2019	OpenAPI-guided request dependencies and stateful fuzzing	Executable sequences and failure observations; specification and reachable state constrain testing
GraphQLer [3] 2026 version	Schema dependency graph and targeted operation chains	Coverage and validated cases on stated targets; metrics depend on the evaluation subset
PentestGPT [4] 2024	LLM modules for testing workflow assistance	Task completion on benchmark targets; not a general detection-accuracy measure
Bulut et al. [5] 2022	Offensive-security-informed prioritization	Case-study evidence for ranking; transfer to another environment needs evaluation

The studies should not be ranked by placing their headline numbers in a single accuracy column. They address different tasks, operate with different information, and use different success criteria. The practical comparison is therefore about complementary roles and missing evidence, rather than a universal best tool.

2. Stateful REST and GraphQL testing

RESTler [2] uses an API specification to identify dependencies between requests and generate sequences in that exercise service state. This addresses a limitation of treating each request as independent: some operations require values created by earlier operations. The work links generated sequences and runtime feedback to deeper exploration and fault discovery. Its reported results apply to its tested services and

configurations, not to every possible REST deployment.

GraphQLer [3] constructs a typed dependency graph from schema introspection and generates operation chains, including cross-user replay and post-delete checks. The reviewed version reports mean Positive Coverage of 85.52% on an 11-public-API coverage subset, while its wider evaluation includes 21 Study Chen et al. [1] 2018 RESTler [2] 2019 Method and input Empirical CORS analysis; browser and site behavior OpenAPI-guided request dependencies and stateful fuzzing Evidence and boundary Trust and configuration weaknesses; interpretation depends on browser semantics Executable sequences and failure observations; specification and reachable state constrain testing

APIs. These denominators must remain separate. Positive Coverage is not the proportion of all vulnerabilities detected. The work supports examining sequences and identities together, but its reported outcomes remain conditional on the evaluated targets and test oracles.

The comparison suggests that requesting syntax alone is insufficient for some auditing goals. State, resource ownership, and the order of operations can be part of the condition being tested. A useful integration should therefore preserve the sequence that led to a finding, rather than retaining only the final response. This is a design inference from the reviewed approaches.

3. Language models in testing workflows

Deng et al. [4] report that language models can assist with tools, output interpretation, and subsequent actions, while struggling to retain the overall testing context. PentestGPT uses interacting modules to address that workflow problem. The reported benchmark improvement concerns task completion relative to a baseline model. It should not be rewritten as a percentage of vulnerabilities detected or as evidence that autonomous operation is reliable in every environment.

For the proposed auditor, a defensible starting point is to let a model explain already collected

observations and suggest review steps. The model should identify which evidence supports each statement. Any proposed action should remain subject to the configured target scope and execution policy. These are design recommendations, not validated features of the student project.

4. Prioritization and remediation decisions

Bulut et al. [5] argue that CVSS alone has limitations for prioritization and proposes an approach inspired by offensive security practice. Their work includes a real-world client case study and examines machine learning for automating the process. It brings attention to the difference between describing a vulnerability and deciding its operational priority. Its evidence does not establish that one ranking strategy transfers unchanged to every organization. In this review, prioritization is treated as a downstream decision. A model cannot compensate for an incorrect underlying finding simply by assigning it a plausible rank. The proposed project should record the basis of a priority decision and allow missing environmental information to remain unknown. An unexplained numeric score would conceal those assumptions rather than resolve them.

V. CROSS STUDY SYNTHESIS

The first shared issue is the meaning of success. Finding an endpoint, reaching application logic, causing an error, completing a testing task, and verifying unauthorised access are different achievements. A review that merges them into one performance measure would obscure what the evidence supports. For a future auditor, these outcomes should appear separately in evaluation and reporting.

The second issue is context. The reviewed work makes context relevant in different ways: browser trust, request dependencies, user identity, conversation state, and organizational priorities. Our synthesis is that an audit record should make the relevant context recoverable. This does not require storing every interaction indefinitely, but it does require retaining enough information to explain and reproduce a consequential finding.

The third issue is the relationship between deterministic checks and AI assistance. Protocol parsing provides explicit conditions that can be inspected. A model can help communicate or reason over those conditions, but its explanation introduces another output that must be checked. Combining the two should preserve a clear boundary between measured observations and generated interpretation.

These conclusions motivate a layered design, but the reviewed studies do not directly compare that complete design against alternatives. Its benefits remain hypotheses for later implementation and controlled evaluation. Table II turns these hypotheses into specific questions without presenting planned work as completed research.

VI. RESEARCH GAPS AND DESIGN IMPLICATIONS

Table 2. Questions for a future AI Security Auditor evaluation

Gap from synthesis	Design implication	Evaluation question
Incomparable outcomes	Separate coverage, verified findings and task success	What does each denominator count?
Incomplete context	Preserve identity, state and request sequence	Can another reviewer reproduce the finding?
Uncertain interpretation	Link explanations to evidence; retain unknowns	Which claims lack direct support?
Model contribution unclears	Compare the same workflow with and without AI	Does assistance improve correctness or review time?
Ranking assumptions hidden	Record environmental inputs and rationale	Does ranking match a stated remediation objective?

These questions are deliberately narrower than asking whether an auditor “works”. They permit

failures to be assigned to the relevant stage. For example, an accurate explanation cannot offset an undiscovered endpoint, while broad endpoint discovery cannot demonstrate that an access-control check was correct. Reporting stage-specific outcomes would make trade-offs easier to interpret. An Evidence traceability

Each future finding should have an identifier, the observed condition, the rule or test that produced it, relevant redacted evidence, and a validation of status. Explanatory prose should refer back to this record. This proposed structure would let a reviewer challenge the inference without having to reconstruct the entire scan from a final narrative report.

A useful status distinction is observed in configuration, suspected weakness, and validated weakness. The transition between these states should require additional evidence, not simply stronger wording. A dismissed finding should retain its reason so that later revisions can be checked for recurring mistakes. The usefulness of this reporting design would still need user evaluation.

2 Authentication and state coverage

A future evaluation should specify which identities and permissions were available, which resources were created during testing and whether the environment was reset between runs. Otherwise, a missed finding might reflect inaccessible functionality rather than an inadequate test rule. Coverage should be reported against an explicit inventory or schema where one exists, with unknown coverage acknowledged elsewhere.

The proposed project should also separate discovery from intrusive validation. A read-only inspection and a state-changing test have different operating assumptions. Scope controls, request budgets, and cleanup behaviour belong in the testing configuration. These are practical implementation recommendations for authorised testing; this review reports no scans of external targets.

3. Evaluating AI assistance independently

To determine whether an AI component adds value, a later study could compare identical observations processed by a deterministic report generator and by a model-assisted generator. Reviewers could assess factual support, remediation relevance, and time required to reach a decision. Repeated runs would help expose output variability. No result from such a comparison is claimed here.

Model name, version, prompt, supplied evidence, and generation settings should be recorded when feasible. Evaluation of examples used to develop prompts should be separated from examples used to assess them. Incorrect suggestions need explicit analysis: a fluent explanation can still omit prerequisites, overstate impact, or recommend a change that does not address the observed condition.

4. Reproducible and useful evaluation

A benchmark for the project should contain both vulnerable and non-vulnerable cases, with the expected outcome and its justification recorded before scoring. Similar variants should test whether a rule recognises the security condition rather than a filename or keyword. Failures should be counted even when they contradict the intended design. These recommendations describe future work, not experiments undertaken for this review.

For deployment of usefulness, evaluation should also include report comprehension and remediation verification. A finding is more actionable when the developer can identify the affected component and confirm whether a subsequent change resolves the condition. The study design should specify whose judgement defines success and how disagreements are handled, rather than assuming every generated recommendation is useful.

Limitations of the Review

The five-study sample is small and purposive. It covers selected technical themes relevant to a web and API auditor, not the full literature on security testing or artificial intelligence. The source-selection process can favour accessible and well-known work. No claim is made about publication of prevalence,

historical trends, completeness of coverage, or the absence of alternative methods.

The synthesis uses qualitative comparisons because the reviewed studies have different targets, baselines, and outcome definitions. Their results were not independently reproduced.

The review therefore cannot establish comparative tool superiority or estimate a pooled detection rate. Statements about individual results must be read with the scope given in the cited source.

Two studies are cited using explicitly identified arXiv versions [3], [5]. Later revisions may alter their methods or results. The protocol documents are also cited by version or access date [6]–[9]; they clarify the semantics considered here rather than certify the behaviour of a particular deployment. These choices make the review traceable while limiting its generality.

The implications for AI Security Auditor are proposed requirements derived from the comparison. The project is not evaluated in this article, and the existence of a requirement does not imply that it has been implemented. A subsequent research paper would need a specified implementation, independent evaluation of cases and reproducible evidence before making performance claims.

VII. CONCLUSION

This focused review finds complementary roles for protocol-aware inspection, stateful API testing, AI-assisted workflow support, and contextual prioritization. The selected studies provide reasons to preserve application state and testing context, but they do not provide interchangeable measures of security effectiveness. The central design implication is that an auditor should keep observations, validation decisions, explanations and priorities distinguishable and traceable.

For the AI Security Auditor project, the next research step is a bounded evaluation of those stages, including negative cases and unsupported model statements. The review supplies the rationale for that

work. It does not claim improved accuracy, successful deployment, or a completed autonomous auditing system.

AI Assistance Disclosure

Generative AI assisted literature organisation, synthesis and manuscript drafting. Author verification of the cited sources, interpretation, and final text is required before submission.

REFERENCES

1. J. Chen, J. Jiang, H. Duan, T. Wan, S. Chen, V. Paxson, and M. Yang, "We Still Don't Have Secure Cross-Domain Requests: An Empirical Study of CORS," in 27th USENIX Security Symposium, 2018, pp. 1079–1093. Available: <https://www.usenix.org/conference/usenixsecurity18/presentation/chen-jianjun>
2. V. Atlidakis, P. Godefroid, and M. Polishchuk, "RESTler: Stateful REST API Fuzzing," in IEEE/ACM 41st International Conference on Software Engineering, 2019, pp. 748–758. doi: 10.1109/ICSE.2019.00083. Author project: <https://github.com/microsoft/restler-fuzzer>
3. O. Tsai et al., "GraphQLer: Enhancing GraphQL Security with Context-Aware API Testing," arXiv:2504.13358v2, Aug. 2026. Available: <https://arxiv.org/abs/2504.13358v2>
4. G. Deng et al., "PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing," in 33rd USENIX Security Symposium, 2024, pp. 847–864. Available: <https://www.usenix.org/conference/usenixsecurity24/presentation/deng>
5. M. F. Bulut, A. Adebayo, D. Sow, and S. Ocepek, "Vulnerability Prioritization: An Offensive Security Approach," arXiv:2206.11182v1, Jun. 2022. Available: <https://arxiv.org/abs/2206.11182v1>
6. OpenAPI Initiative, "OpenAPI Specification," version 3.1.0, Feb. 2021. Available: <https://spec.openapis.org/oas/v3.1.0.html>. Accessed: Sep. 14, 2026.
7. GraphQL Foundation, "GraphQL Specification," October 2021 edition. Available:

<https://spec.graphql.org/October2021/>.

Accessed: Sep. 14, 2026.

8. A. Barth, "The Web Origin Concept," RFC 6454, Dec. 2011. doi: 10.17487/RFC6454. Available: <https://www.rfc-editor.org/rfc/rfc6454>
9. WHATWG, "Fetch Standard," Living Standard, section on the CORS protocol and credentials. Available: <https://fetch.spec.whatwg.org/#cors-protocol-and-credentials>. Accessed: Sep. 14, 2026.