SecContext-Fin: Structuring LLM-Based Red/Blue Security Tools for the Secure Software Development Lifecycle

Juka Francis Pereira Gonçalves
Federal University of Lavras
São Sebastião do Paraíso, Minas Gerais, Brazil
juka.goncalves@estudante.ufla.br

Maria Júlia Pedroso Pimenta
Federal University of Lavras
São Sebastião do Paraíso, Minas Gerais, Brazil
maria.pimenta3@estudante.ufla.br

Bento Rafael Siqueira
Federal University of Lavras
Lavras, Minas Gerais, Brazil
bento.siqueira@ufla.br

Samira Santos da Silva
Federal University of Lavras
São Sebastião do Paraíso, Minas Gerais, Brazil
samirasilva@ufla.br

Diego Saqui
Federal University of Lavras
São Sebastião do Paraíso, Minas Gerais, Brazil
diego.saqui@ufla.br

Alysson Alexander Naves Silva
Federal University of Lavras
São Sebastião do Paraíso, Minas Gerais, Brazil
alysson.naves@ufla.br

Abstract

LLM-based security assistants can support vulnerability analysis, exploitation reasoning, and mitigation planning, but outputs detached from code evidence and review gates are difficult to reuse safely. SecContext-Fin is an evidence-grounded framework that turns a prioritized vulnerability input into auditable red-team and blue-team analyses through four responsibilities: Evidence Framing, Context Construction, Red/Blue LLM Analysis, and Audit and Learning. We instantiate it with FinSim and a preliminary table of 50 candidate findings produced with LLM assistance. The authors selected a Top-10 and converted it into Security Context Cards, prompts, and registry records. A nine-execution pilot reports after-the-fact overlap for blind repository exploration and mean author scores of 2.58 and 2.99 (1–3 scale) for targeted-generic and grounded prompts, respectively. The descriptive pilot does not validate vulnerabilities or replace independent reviewers; it shows how explicit evidence, uncertainty, and provenance can turn prompt-driven output into inspectable SSDLC review material.

Keywords

Large language models, AI-assisted security, Software security, Vulnerability analysis, Evidence-grounded prompting, Red-team and blue-team analysis, Financial software, Auditability

1 Introduction

Motivation. Financial software handles authentication, transaction processing, token management, and personal or payment-related data. Vulnerable code may expose confidential data, compromise APIs, enable fraud, or weaken trust in digital services [10, 15]. These risks align with recurring Web application classes such as injection, broken access control, cryptographic failures, and security misconfiguration [4]. At the same time, LLM-based programming and security assistants are becoming part of secure software development, raising questions about how AI-assisted analysis can remain traceable, reviewable, and grounded in evidence throughout the Secure Software Development Lifecycle (SSDLC) [12]. For security-oriented AI-assisted development, the central problem is not only whether an LLM can mention plausible weaknesses, but whether its outputs can be traced back to concrete code evidence before they affect design, implementation, testing, or release decisions [13].

Problem. Red-team and blue-team security tools require different reasoning over the same evidence. Red-team analysis emphasizes exploitability, preconditions, and impact, while blue-team analysis emphasizes mitigation, validation, and prioritization [7, 8]. If such tools are built only around free-form prompts, their outputs may be difficult to compare, reproduce, or audit. Preliminary vulnerability reports can provide useful evidence, but they are not final ground truth: findings may be duplicated, partial, or dependent on code-level confirmation.

Proposal. This paper presents *SecContext-Fin*, an evidence-grounded framework and artifact package for AI-assisted red/blue security analysis in financial software. The proposal does not replace vulnerability detectors, human reviewers, or secure-development practices. Instead, it defines technology-independent building blocks that transform a prioritized Top-10 set of preliminary findings into *Security Context Cards*, route these cards to complementary red/blue prompt templates, and retain an audit registry that connects FinSim findings, prompts, LLM responses, and human validation. In this study, the preliminary findings come from an LLM-assisted vulnerability analysis of the FinSim codebase, not from an independently curated benchmark. Thus, SecContext-Fin is positioned as reusable support for security tools in AI-assisted development, not as a claim that one model or prompt is sufficient for vulnerability confirmation.

Contributions. The paper makes the following contributions:

- an evidence-grounded framework with four responsibilities for AI-assisted red/blue security analysis with explicit traceability from evidence to prompt, response, and review decision;
- concrete artifacts: a Top-10 vulnerability input, Security Context Cards, red/blue prompt templates, and a prompt-response registry;
- a FinSim instantiation based on 50 preliminary findings produced by an LLM-assisted vulnerability analysis; and
- a pilot with three prompting conditions, each repeated three times, comparing blind, targeted-generic, and context-grounded

inputs through six reviewability dimensions with archived prompt/response records.

2 Background

Financial Software Vulnerabilities. Financial applications combine security-sensitive assets with Web, API, and service integration concerns. Prior work on fintech cybersecurity discusses phishing, API compromise, data leakage, and fraud-oriented attacks [10], while API-centric financial systems face risks from weak authentication, misconfigured endpoints, and insufficient access control [15]. The OWASP Top 10 provides a vocabulary for recurring Web weaknesses such as injection, authentication failures, security misconfiguration, and vulnerable components [4], categories visible in the FinSim findings used in this paper.

Evidence, Testing, and Validation. Testing and vulnerability analysis are evidence-oriented activities: tests can reveal faults, but cannot prove their absence; candidate findings require confirmation; and validation depends on expected behavior, observed behavior, and risk reduction [5, 16]. SecContext-Fin adopts the same stance: a finding becomes useful for LLM-supported security analysis only when its evidence, assumptions, and validation needs are explicit.

Secure-Development Frameworks. The Secure Software Development Framework (SSDF) organizes secure development around preparing the organization, protecting software, producing well-secured software, and responding to vulnerabilities [12]. SecContext-Fin uses this view to organize responsibilities, artifacts, information dependencies, and review points. The terminology is informed by software-architecture research on components, relationships, constraints, and design decisions [6, 9, 14], but SecContext-Fin does not propose a complete architectural blueprint. Its narrower scope is to organize evidence, cards, prompts, review records, and knowledge reuse for AI-assisted red/blue security analysis.

3 Related Work

AI-Assisted Security. Recent work explores transformer-based vulnerability detection at design time [3], code-property-graph representations with neural models [17], automated pentesting and remediation [8], and ontology-assisted vulnerability evaluation [7]. These studies show that AI can help identify, explain, or prioritize security issues. The gap addressed by SecContext-Fin is complementary: security answers must be grounded in source-code evidence, expose assumptions, support validation, and remain auditable before they influence secure-development decisions.

LLM-Integrated Systems. Software architecture treats architecture as components, relationships, constraints, and design decisions [6, 9, 14]. Recent work applies related ideas to LLM-integrated and agent-based systems [1, 11]. SecContext-Fin uses this literature as conceptual support, but its contribution is narrower: an artifact package for AI-assisted red/blue security analysis, starting from a prioritized Top-10 vulnerability input and ending in reviewable cards, prompts, responses, and registry records.

4 SecContext-Fin Framework

Framework Scope. SecContext-Fin turns prioritized vulnerability findings into auditable red/blue LLM security-analysis tasks. Its scope is the analysis-support layer around the target system, leaving model providers, IDEs, CI tools, dashboards, and deployment choices open. This separation is intentional: the framework specifies what evidence and review responsibilities a tool should preserve, while concrete implementations may use a chat interface, an IDE extension, a CI workflow, or future red/blue bots.

Evidence Framing. Figure 1 is read from left to right. A security analyst starts from the FinSim codebase and a prioritized Top-10 set obtained from a preliminary LLM-assisted vulnerability analysis. Evidence Framing scopes the system, the candidate finding, and the security question; it does not confirm final vulnerability correctness.

Context Construction. Context Construction connects each Top-10 item to source-code evidence such as files, line ranges, snippets, and vulnerability class. The Evidence Base stores candidate facts, the Card Builder normalizes them, and the Security Context Cards expose fields such as `finding_id`, asset, `code_evidence`, weakness, severity, assumptions, mitigation, and validation.

Red/Blue LLM Analysis. Prompt Orchestration uses the cards to instantiate structured red/blue prompts. The red-team role focuses on exploitability, attacker preconditions, impact, and missing evidence; the blue-team role focuses on mitigation, prioritization, validation, and regression tests. These are logical tool roles, not mandatory bots. Both roles consume the same card so that disagreements or complementary recommendations can be reviewed against the same evidence base, instead of being judged as unrelated free-form answers.

Audit and Learning. The Audit Registry records the finding, card, prompt, response, prompting condition, and reviewer status. It is the framework's traceability mechanism, keeping evidence, prompt variant, LLM output, and reviewer decision in one auditable chain. The Human Review Gate checks whether the LLM preserved evidence, introduced unsupported claims, or produced actionable validation steps before any review package influences triage, remediation, or tests. The resulting SSDLC review package is the handoff artifact: it keeps accepted evidence, role-specific reasoning, mitigation tasks, validation checks, and rejected or uncertain claims. Knowledge Reuse stores only reviewed artifacts; the dashed path denotes reviewed rather than blind reuse. Constraints and variation points preserve evidence and allow providers or integrations to vary without changing the logic.

5 FinSim Case Study

FinSim and the Top-10 Input. We instantiate SecContext-Fin with FinSim, an open-source financial simulator from Carnegie Mellon University Software Engineering Institute [2]. FinSim models merchants, credit-card processors, and banks, and includes security-relevant concerns such as token handling, database access, configuration, and dependencies. The analyzed upstream snapshot corresponds to commit `4d7b10838a98`. The archived workbook contains 50 contiguous candidate rows, each recording a file/line, code snippet, detection rule, OWASP mapping, likelihood, impact, severity, confidence, mitigation, and status. It preserves neither the exact model/prompt that generated the original LLM-assisted table nor an independent manual or SAST validation.

Table 1: Comparison with representative approaches; "Partial" denotes indirect support.

Approach	Evidence	R/B roles	Human validation	Auditability	SSDLC link
Edit-time/graph detectors [3, 17]	Code or graphs	No	Partial	Model output	Detection
PenHeal [8]	System/external knowledge	Pentest/remediation	No required gate	Partial	Remediation
CVE-LLM [7]	CVEs and ontologies	No	Expert-facing	Partial	Evaluation
LLM/agent architectures [1, 11]	System context	Partial	Governance-level	Conceptual	General
SecContext-Fin	Files, lines, cards	Explicit R/B	Required gate	Prompt/response registry	Review gates

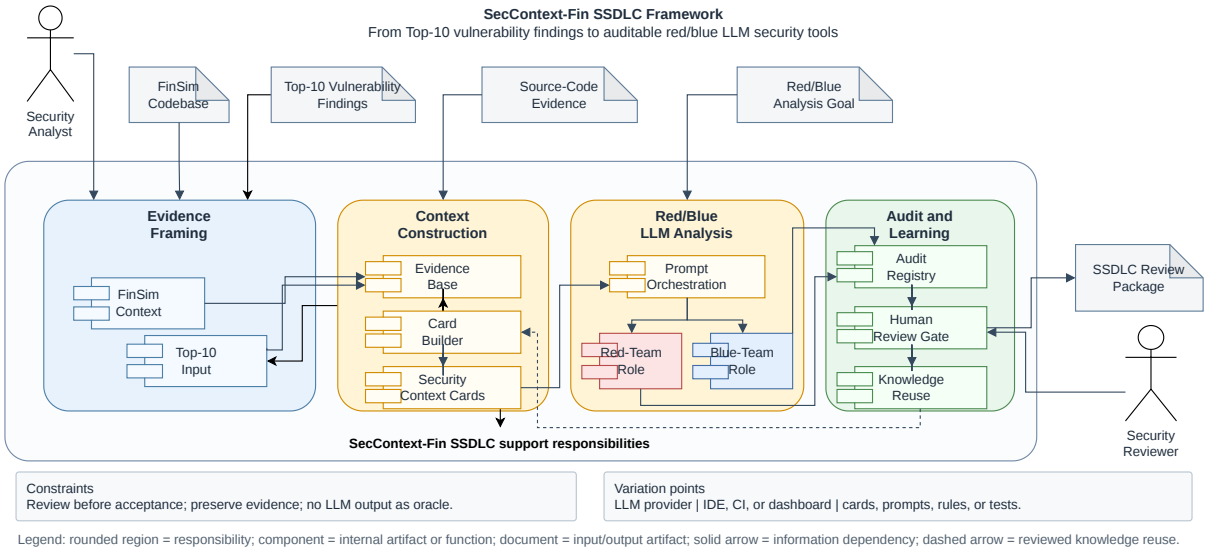

Figure 1: SecContext-Fin framework organized by responsibility. Rounded regions denote logical responsibilities; component boxes denote internal artifacts or functions; document shapes denote input/output artifacts; arrows denote information dependencies and reviewed knowledge reuse.

Table 2: Controlled Top-10 FinSim input used to instantiate SecContext-Fin.

Rank	Finding family	Severity	Use
1	SQL injection	Critical	R/B
2–3	Hardcoded secrets	Critical	Blue
4	Browser token storage	High	R/B
5	Token logging	High	Blue
6–8	Misconfiguration	High	Blue
9	Outdated dependency	High	Blue
10	Token injection	High	R/B

Table 3: Inputs and descriptive pilot results across three runs per condition.

Cond.	Input	Descriptive result
C1	Repository only	5/10 Top-10 items surfaced in all three runs; TOP-09 surfaced in 0/3; not rubric-scored.
C2	ID, title, file	Mean overall reviewability: 2.58/3; evidence was often summarized and assumptions mixed with facts.
C3	Security Context Card	Mean overall reviewability: 2.99/3; files/lines and uncertainty were more consistently preserved.

The workbook's separate Top-10 sheet consolidates related rows by finding family and affected asset and supplies a manual rank; it encodes no automatic ranking formula. Critical groups precede High groups, but remaining ordering reflects author curation. We therefore treat the Top-10 as a prioritized input artifact, not a validated vulnerability benchmark.

Cards, Prompts, and Pilot. For each Top-10 item, we created a Security Context Card with the affected asset, source evidence, weakness, severity, assumptions, mitigation, and validation fields. We then instantiated red-team prompts about attack paths and missing preconditions and blue-team prompts about mitigation and secure-development checks. C1 gave three agents only the repository; C2 gave three agents the Top-10 identifier, title, and file; C3 gave three agents the corresponding card. Exact experimental prompts, raw responses, hashes, and associations are archived. C1 is reported only as post-hoc overlap because its repository-level discovery task was not aligned to the same tuples as C2 and C3.

The author team scored C2 and C3 with six dimensions. Traceability measures links to findings, files, and lines; specificity measures target-level detail; evidence preservation measures retention of supplied source facts; hallucination control measures separation of confirmed, inferred, and unconfirmed claims; mitigation measures concrete repair guidance; and validation measures executable checks. Each dimension uses 1 (absent or unsupported), 2 (partial or generic), or 3 (explicit and actionable). A run's overall score is the arithmetic mean of its six dimensions, and a condition score averages its three runs.

TOP-09 illustrates the difference. C2 stated that Django 1.9.1 was the runtime and accumulated known vulnerabilities. C3 preserved only the file's comment and documentation links as evidence, marked the installed runtime, end-of-life status, and specific CVEs as unconfirmed, and recommended checking the dependency before upgrading. This improves claim auditability, not vulnerability correctness.

Execution and Review Provenance. The original execution record did not retain the LLM model/version, decoding parameters, execution dates, or context-window configuration; exact generative reproduction is therefore not claimed. The authors applied the rubric without blinding. No independent security-professional panel, per-reviewer score table, or inter-rater agreement statistic was collected.

6 Discussion and Conclusion

Discussion. SecContext-Fin replaces isolated prompts with a reviewed artifact chain linking a Top-10 finding, card, red/blue analysis, registry, human review, and reuse. Shared evidence supports red-team exploitability reasoning and blue-team mitigation, validation, and regression tests. The higher C3 mean (2.99 versus 2.58 for C2) is descriptive evidence that the cards improved the author team's ability to trace these outputs; it is not a statistical or independent validation of security accuracy.

Practically, SecContext-Fin supports rather than replaces security judgment. LLMs organize evidence and draft red/blue analyses; reviewers decide which claims are supported and which mitigations are actionable. This separation keeps security reasoning inspectable and revisable as code, evidence, and organizational priorities evolve. Accordingly, SecContext-Fin shifts evaluation from isolated LLM answers to provenance-preserving workflows, connecting prompt quality to reproducible reasoning, human review, and actionable SSDLC decisions.

Open Science and AI Use. The replication package provides the study inputs, prompts, responses, hashes, registries, evaluation scores, and reproduction guidelines. AI tools supported grammar revision, prompt drafting, table organization, and pilot-output comparison; the authors reviewed all security interpretations and paper claims.

Threats to Validity. One system, an unvalidated 50-finding table, a curated Top-10, and nine executions limit the study. Model and decoding metadata were not retained, and the authors applied the post-hoc rubric without blinding or agreement measurement. These limits preclude generalization. Future work should add a second system, manual/SAST validation, and 3–5 blinded professionals with per-reviewer scores and agreement analysis.

Conclusion. This paper proposed SecContext-Fin as an evidence-grounded framework for turning prioritized FinSim findings into auditable red/blue LLM security analysis. Its contribution is not an isolated LLM answer, but a compact artifact package – cards, prompts, registry, and reviewed reuse – that turns preliminary findings into SSDLC review material. The pilot supports this feasibility claim while leaving vulnerability validation and effectiveness across systems to a larger independent study.

Artifact Availability

The replication package is available at https://github.com/californi/seccontext-fin. It includes the FinSim snapshot, 50-finding workbook, curated Top-10, cards, exact C1–C3 prompts, nine raw responses, hashes and registries, evaluation table, and reproduction guidelines. Documentation and derived artifacts use CC BY 4.0, scripts use MIT, and FinSim retains its upstream license.

Acknowledgments

The authors acknowledge Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), Brazil, for support through projects CIN II APQ-06513-25 and CEX-BIS-00192-25.

References

[1] Alessio Bucaioni, Martin Weyssow, Junda He, Yunbo Lyu, and David Lo. 2025. A Functional Software Reference Architecture for LLM-Integrated Systems. In *ICSA-C*. doi:10.1109/ICSA-C65153.2025.00006

[2] Carnegie Mellon University Software Engineering Institute. 2019. FinSim: Financial Simulator. GitHub repository. https://github.com/cmu-sei/finsim Analyzed commit 4d7b10838a98; accessed 2026-06-25.

[3] Aaron Chan, Anant Kharkar, Roshanak Zilouchian Moghaddam, Yevhen Mohylevskyy, Alec Helyar, Eslam Kamal, Mohamed Elkamhawy, and Neel Sundaresan. 2023. Transformer-based Vulnerability Detection in Code at Edit-Time: Zero-shot, Few-shot, or Fine-tuning? arXiv:2306.01754 [cs.CR] https://arxiv.org/abs/2306.01754

[4] OWASP Foundation. 2021. OWASP Top 10 – 2021: The Ten Most Critical Web Application Security Risks. Available at: https://owasp.org/Top10/.

[5] Phyllis G. Frankl and Elaine J. Weyuker. 2000. Testing Software to Detect and Reduce Risk. *Journal of Systems and Software* 53, 3 (2000), 275–286. doi:10.1016/S0164-1212(00)00018-2

[6] David Garlan and Mary Shaw. 1994. *An Introduction to Software Architecture*. Technical Report CMU-CS-94-166. Carnegie Mellon University.

[7] Rikhiya Ghosh, Hans-Martin von Stockhausen, Martin Schmitt, George Marica Vasile, Sanjeev Kumar Karn, and Oladimeji Farri. 2025. CVE-LLM: Ontology-Assisted Automatic Vulnerability Evaluation Using Large Language Models. arXiv:2502.15932 [cs.CL] https://arxiv.org/abs/2502.15932

[8] Junjie Huang and Quanyan Zhu. 2024. PenHeal: A Two-Stage LLM Framework for Automated Pentesting and Optimal Remediation. arXiv:2407.17788 [cs.CR] https://arxiv.org/abs/2407.17788

[9] Anton Jansen and Jan Bosch. 2005. Software Architecture as a Set of Architectural Design Decisions. In *WICSA*. 109–120. doi:10.1109/WICSA.2005.61

[10] Paulin Kamuangu. 2024. A Review on Cybersecurity in Fintech: Threats, Solutions, and Future Trends. *Journal of Economics Finance and Accounting Studies* 6 (02 2024), 47–53. doi:10.32996/jefas.2024.6.1.5

[11] Qinghua Lu, Liming Zhu, Xiwei Xu, Zhenchang Xing, Stefan Harrer, and Jon Whittle. 2024. Towards Responsible Generative AI: A Reference Architecture for Designing Foundation Model Based Agents. In *ICSA-C*. 119–126. doi:10.1109/ICSA-C63560.2024.00028

[12] National Institute of Standards and Technology. 2022. *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. Technical Report NIST Special Publication 800-218. National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-218

[13] OWASP Foundation. 2025. OWASP Top 10 for Large Language Model Applications. https://genai.owasp.org/llm-top-10/. Accessed: 2026-06-28.

[14] Dewayne E. Perry and Alexander L. Wolf. 1992. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes* 17, 4 (1992), 40–52. doi:10.1145/141874.141884

[15] Piyush Ranjan and Mohammad Haider. 2024. API Security Challenges and Risk Mitigation in Fintech Applications. International Journal of Global Innovations and Solutions. doi:10.21428/e90189c8.43a4136c

[16] Debra J. Richardson, Stephanie Leif Aha, and T. Owen O'Malley. 1992. Specification-Based Test Oracles for Reactive Systems. In *ICSE*. 105–118. doi:10.1145/143062.143100

[17] Amanpreet Singh Saimbhi. 2025. Enhancing Software Vulnerability Detection Using Code Property Graphs and Convolutional Neural Networks. In *ICCCIT*. 435–440. doi:10.1109/icccit62592.2025.10928033