Journal of Information Systems Engineering and Management
2026, 11(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

Beyond Anomaly Alerts: An Independent Assessment of ZeroTrust SecOps AI and Its Evidence-Governed Response Workflow

Sanika Satish Lad[1], Afzal Siraj Khan[2], Kush Pulindra Patel[3]

[1] *Master of Data Science, The University of Western Australia, Perth, Australia.*

[2] *Department of Artificial Intelligence and Machine Learning, Thakur College of Engineering and Technology, Mumbai, Maharashtra India.*

[3] *Master of Data Analytics Engineering, Northeastern University, Boston, MA, USA.*

ARTICLEINFO

Received: 03 Aug 2026

Revised: 23 Sept 2026

Accepted: 07 Oct 2026

ABSTRACT

An unfamiliar Linux command may indicate either compromise or an authorized operational change; novelty alone cannot settle the distinction. This paper presents an independent, repository-based assessment of ZeroTrust SecOps AI, developed by Arya Brijesh Tiwari, with particular attention to the controls between anomaly scoring and security action. The review authors obtained and ran the application locally, observing successful startup and usable backend/dashboard workflows; this smoke test is distinct from a security benchmark or a live cloud deployment. We examine the public architecture and supplied verification record using a four-part rubric covering identity and access boundaries, baseline governance, evidence integrity and cloud-response safeguards. The implementation correlates Linux audit records, compares execution vectors with operator-approved host baselines, retains an explicit unscored state when evaluation is unjustified, signs evidence with Ed25519 and guards optional AWS containment. The project verification report records 78 passing Python tests, ten frontend-core checks and syntax transpilation of 29 TypeScript/TSX files; those figures were not independently reproduced as a complete suite by the reviewers. Relative to the stand-alone anomaly-detection studies examined, ZeroTrust SecOps AI provides a particularly cohesive detection-to-decision workflow. Its architectural breadth is a strength under the stated rubric, while comparative detection accuracy, large-scale performance and live containment remain unmeasured.

Index Terms— anomaly detection, zero trust, Linux audit, security operations, human-in-the-loop baselines, digital signatures, cloud containment, incident response.

I. INTRODUCTION

Zero-trust thinking begins with a restraint: a resource should not be trusted merely because it resides on a familiar network or belongs to a known organization. NIST SP 800-207 translates that restraint into identity- and resource-centered access decisions [1]. A similar restraint is needed after a host-based detector observes an unfamiliar command. The fact that an execution differs from historical behavior is useful evidence, but it does not by itself establish intent, impact, or authorization.

Consider a Linux server that has routinely executed a small set of maintenance commands. A new Python script appears after a deployment. A novelty detector may assign the execution a low similarity score; the same observation could also arise when an intruder introduces a script. If the system equates novelty with compromise, routine changes generate avoidable incidents. If it automatically absorbs all new activity into the baseline, hostile behavior may eventually appear normal. The operational problem lies between these extremes: the detector can prioritize an event, but a separate process must decide what the observation warrants.

System-call anomaly detection has a long research history. Warrender et al. compared models of normal call sequences and showed that useful deviations can be captured without requiring the most complex statistical model

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

[2]. DeepLog subsequently learned normal log sequences with recurrent neural networks [3], while LogBERT [4], LAnoBERT [5] and NeuralLog [6] advanced representation learning for log-based anomaly detection. These studies provide a strong foundation for discovering unusual patterns. A recent survey also identifies reproducibility, evaluation design and explainability as recurring difficulties in log-anomaly research [7]. This review asks a related question: how convincingly does the examined implementation handle a novelty finding after a detector produces it?

The publicly available ZeroTrust SecOps AI implementation, developed by Arya Brijesh Tiwari [13], provides an end-to-end path from audit-record collection to investigation and optional response. The three authors of this manuscript review the implementation independently and do not claim to have designed or developed the software. An agent bound to a declared host identity sends Linux audit records to a FastAPI service. The backend correlates records belonging to one execution, represents the resulting context as a vector and compares it only with behavior that an operator has reviewed for that host and encoder profile. When the comparison cannot be justified, the event remains unscored. The system signs the original evidence, retains a persistent history, streams bounded event previews to a dashboard and permits cloud containment only after explicit configuration and independent safety checks.

The three manuscript authors contribute an independent repository-based review, not a new detector or the source implementation. Their work comprises a local application smoke test, a documented four-criterion assessment rubric, examination of design and verification claims and a structured comparison with published anomaly-detection approaches. We distinguish features inspected in the artifact, outcomes reported by its existing test suite and behavior actually observed in the reviewers' local run. The software and architecture remain credited to Tiwari [13]. The conclusion concerns the completeness of the governed investigation workflow under these criteria, not a claim of state-of-the-art classification performance.

II. RELATED WORK AND RESEARCH GAP

A. From system-call models to learned log representations

Early host-based methods represented normal execution through short system-call sequences. Warrender et al. examined enumeration, frequency, rule-induction and hidden Markov approaches, highlighting a trade-off between model complexity and the regularity of the observed workload [2]. These methods are useful because they ground anomalies in host behavior rather than requiring an existing attack signature. Their effectiveness nevertheless depends on what is observed, how normality is constructed and whether workload changes resemble attacks.

DeepLog uses an LSTM to model log-event sequences and identify departures from learned patterns [3]. LogBERT adopts self-supervised Transformer learning to capture normal log sequences [4]. LAnoBERT reduces dependence on conventional parsing through masked-language-model objectives [5] and NeuralLog uses representations of raw log messages with a Transformer-based classifier [6]. Their reported evaluations emphasize anomaly-discrimination performance on specific datasets. These results are not directly transferable to the present prototype: its input is correlated Linux execution telemetry and its default representation is feature hashing rather than a trained sequential model.

B. Semantic representations and baseline control

Sentence-BERT showed how sentence embeddings can make cosine-based semantic comparison practical [8]. MiniLM explored compact Transformer representations through knowledge distillation [9]. ZeroTrust SecOps AI exposes an optional Sentence-Transformer path using all-MiniLM-L6-v2, alongside a local deterministic hashing encoder. The two encoders are not interchangeable: one provides a trained semantic representation, while the other produces a repeatable vector from selected features without neural inference. A baseline is therefore bound to the encoder profile that produced its vectors.

Most published anomaly detectors necessarily construct normality from training observations. That is a sound research choice when training data are curated, but it leaves a deployment question: which observations should be trusted when the system is learning from a live host? The reviewed implementation does not solve baseline poisoning by proving that human judgment is infallible. It instead creates an explicit approval boundary, making it

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

possible to inspect who accepted an event as normal and why. This trades operator effort for a more accountable learning process.

C. The missing link between detection and action

A model benchmark usually asks whether a record is correctly labelled. A security operations workflow must additionally ask whether the record is complete, whether its origin is authorized, whether its evidence can be checked and whether a response can be recovered. Digital signatures provide one mechanism for checking retained evidence integrity; Ed25519 offers a compact signature scheme suitable for independent verification [10]. MITRE ATT&CK supplies a vocabulary for behaviors such as Unix-shell and Python execution, but a technique association is not an attribution of malicious intent [11]. Incident-response guidance likewise treats detection, analysis, containment and recovery as related but distinct activities [12].

The project developed by Tiwari [13] brings these controls into one reference implementation. Under the criteria used in this review baseline governance, abstention, evidence integrity and response safeguards its integrated workflow is a notable advantage over evaluating a stand-alone classifier alone. This is a comparison of system scope, not proof of higher F1 score than DeepLog, LogBERT, LAnoBERT, or NeuralLog. A detection-performance ranking would require shared datasets, labels, thresholds and measurement conditions.

III. System Architecture And Threat Model

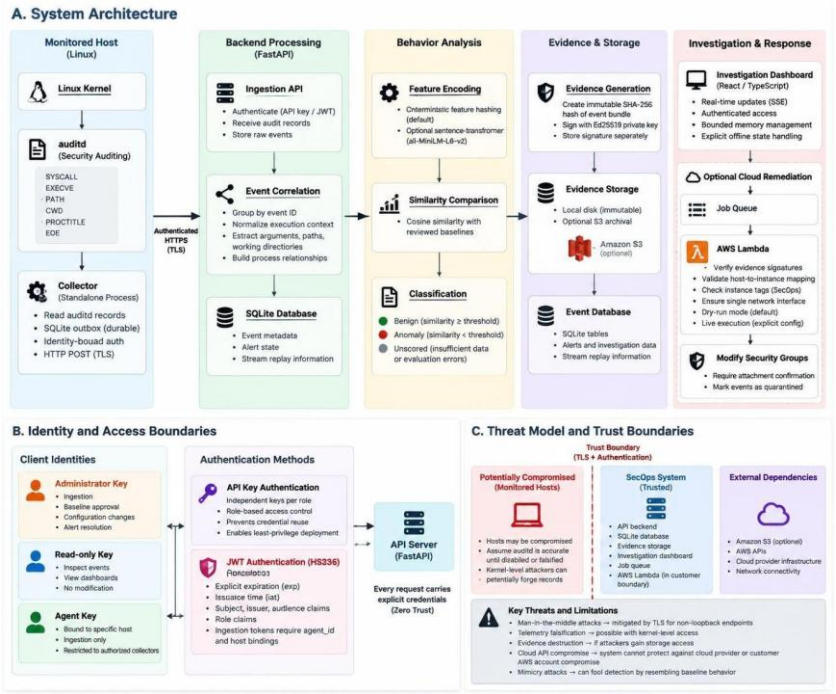


Figure 1: System Architecture of ZeroTrust SecOps AI

A. Processing path and trust boundaries

The reference architecture shown in figure 1 consists of five cooperating elements: a Linux collector, an authenticated ingestion service, a detector and persistence layer, an investigation dashboard and an optional cloud-response handler. The collector reads audit-format records and maintains a SQLite outbox and file cursor. Acknowledged data are removed from the outbox; transient failures lead to retries. Delivery is therefore designed for retry with idempotent server processing, not an unqualified claim of exactly-once transport.

The backend groups SYSCALL, EXECVE, PATH, CWD, PROCTITLE and end-of-event records using their shared audit message identity, including the event serial. The auid field, when present, describes the audit login identity and is not the unique event identifier. Correlation produces an execution context with its source, observed time,

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

executable, arguments and available path/working-directory information. Incomplete or late-arriving records are preserved without inventing missing arguments.

The service persists events, baseline membership, alerts, configuration, stream sequence information and remediation jobs in SQLite. Each finalized event receives a canonical signed evidence snapshot. The dashboard reads persistent history through authenticated REST requests and subscribes to Server-Sent Events (SSE) for timely updates. Browser previews are bounded to 500 entries; complete retained event records remain available through the API. An optional WebSocket endpoint uses short-lived single-use tickets for browser authentication.

Processing chain: audit records → durable outbox → authenticated ingestion → event correlation → host/profile-specific comparison → persistent event and signed evidence → investigation → optional cloud action.

B. Identity, privileges and control plane

Three roles divide the application surface. An ingest credential is restricted to its bound collector and host identity. A read credential can inspect events, health and evidence. An admin credential can additionally approve baseline members, update configuration and manage alert workflow. JWT-based authentication is also supported under explicit configuration. This partition prevents a collector key from becoming a general administrative credential.

The cloud-response path is intentionally separate from ingestion. Enabling remediation requires an administrator-controlled configuration decision, but the current implementation should not be described as requiring a fresh human approval for every individual containment action. Its safety boundary is formed by disabled-by-default configuration, dry-run operation, trusted evidence, target restrictions and verification of the resulting cloud state.

C. Attacker capabilities and residual risks

The threat model considers forged collector identities, replayed input, tampered stored evidence, incomplete audit records, model-service failure, unsafe cloud target selection and lost recovery state. Identity binding, deterministic event fingerprints, signed snapshots, explicit unscored outcomes, target allowlists and preserve-before-mutate ordering address these risks within the tested scope.

The system does not establish the truth of a compromised kernel's audit output. An attacker with sufficient host privilege may suppress telemetry before collection. A signed file is tamper-evident, not physically immutable: an attacker who controls storage may delete it and an attacker who steals the signing key may create apparently valid signatures. A verified security-group attachment does not prove that every established network connection has terminated. The single-process SQLite design also does not provide distributed high availability. These boundaries are material to the security claim.

IV. DETECTION, EVIDENCE AND RESPONSE METHODOLOGY

A. Correlated execution representation

The detector operates on a reconstructed execution, not on each audit line independently. Correlation collects records with the same audit message identifier, extracts supported quoted and hexadecimal fields and retains the original lines. If long argument fragments cannot be reconstructed reliably, the event remains partial and is not given a fabricated score. This matters because an apparently harmless truncated command may differ substantially from the actual invocation.

Let x_e denote the normalized context for event e and let phi_m(x_e) be its vector under encoder profile m. The default profile uses deterministic signed-feature hashing. An optional profile uses a Sentence-Transformer model and, when configured, an external Pinecone index. The current local results concern the default path; live MiniLM/Pinecone performance is not measured here.

B. Reviewed similarity and abstention

For host h and encoder profile m, define the approved baseline as B(h,m). For an eligible new event, similarity is computed against the reviewed vectors:

$$s(e, h, m) = \max_{b \text{ in } B(h,m)} \cos\left(\text{phi}_m(x_e), \text{phi}_m(x_b)\right). \quad (1)$$

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

where e denotes the new event being evaluated, h denotes the host on which the event occurred, and m denotes the encoder profile used to represent the event. B(h,m) represents the set of approved baseline events associated with host h and encoder profile m. The terms xe and xb denote the normalized feature representations of the new event and a baseline event b, respectively. The function $\phi m(\cdot)$ represents the vector encoder associated with profile m, while $\cos(\cdot,\cdot)$ denotes cosine similarity between two vectors. The operator max selects the highest cosine similarity between the new event and all eligible events in the approved baseline.

The decision threshold is tau (0.55 in the documented local default). A scored event is anomalous when $s < tau$; otherwise it is classified as benign. The score measures representational similarity, not the probability of attack. At the boundary, equality with tau is benign under the implemented strict inequality.

Scoring is withheld when the baseline lacks sufficient approved complete examples, when learning mode is active, when the event is partial, or when the detector cannot evaluate it. The status is then unscored and the score is null. The baseline minimum is 20 approved complete events per host/profile by default. Twenty is a configuration requirement, not a statistically established sample-size guarantee.

Learning mode records observations without classifying them against an active baseline. Static mode freezes membership. Adaptive mode allows continued scoring and manual additions, not unrestricted automatic learning. Approval requires an administrator's explicit selection and reason. Earlier unscored events do not retrospectively become benign merely because they are later approved.

C. Evidence with independent verification

The evidence service serializes an original event snapshot as canonical JSON and calculates its SHA-256 digest. It signs the canonical bytes with a persistent Ed25519 private key and makes the corresponding public key available for independent checking. The verifier recomputes the digest and checks the signature rather than trusting a stored verified flag. Baseline membership, later alert decisions and remediation outcomes remain mutable operational state and are not rewritten into the original signed snapshot.

This construction supports detection of unauthorized modification to retained evidence under a trusted-key assumption. It does not prove the original telemetry was complete or authentic at the kernel and it does not replace protected key custody or storage backups. Late audit records are exposed separately instead of altering already signed content.

D. Investigative context and cloud response

A novelty finding may be associated with MITRE ATT&CK context, such as Unix shell (T1059.004) or Python (T1059.006). These mappings help analysts choose questions to ask; they are not claims that an attack has occurred. The dashboard therefore keeps similarity, evaluation state, host identity and evidence visible together.

Optional remediation follows a conservative sequence. It first checks signed evidence and trusted target binding, then applies configured allowlist and managed-tag restrictions. The handler is limited to the supported single-network-interface case. Before any permitted security-group change, it stores the original attachments in DynamoDB for later recovery. After requesting a replacement attachment, it reads the cloud configuration back and records success only if the expected change is observed. S3/KMS-backed evidence and recovery resources are defined in the optional infrastructure configuration. Dry-run mode performs no live AWS mutation. These behaviors have been exercised with mocked cloud clients, not with a completed live containment trial.

V. REVIEW METHOD AND REPORTED VALIDATION

A. Review protocol and evidence boundary

This study uses an artifact-oriented review rather than a newly developed detection model. Four questions guide the examination: RQ1, are telemetry identities and privileges constrained? RQ2, is normal behavior governed through reviewed baselines and explicit abstention? RQ3, can an investigator verify original evidence independently? RQ4, does the optional cloud response preserve safe defaults and a recovery path? The authors assess source-described mechanisms, reported tests and local runtime observations separately. They did not conduct a new labelled attack-detection experiment.

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

Three evidence classes are kept distinct. Observed denotes behavior seen during the authors' local application run; reported denotes outcomes in the developer-provided verification record; and designed denotes functionality visible in code or configuration but not exercised end to end by the review team. A positive judgment about a designed safeguard does not imply it was validated on live AWS infrastructure.

B. Independent local application run

The reviewers used the public ZeroTrust SecOps AI repository [13] as their sole software artifact, configured the local Python and Node.js environments, started the FastAPI backend and React/Vite dashboard and observed that the application launched and the interface was usable. This is an independent functional smoke test of the local application, not a reproduction of the complete published test suite. It supports the practical claim that the supplied application can be started and inspected on a development machine. It does not establish that live Linux audit collection, semantic embedding services, or AWS containment were operating during the run.

The local run was not accompanied by a timestamped, archived test log, machine specification, or formal navigation checklist. Consequently, this review does not assign numerical availability, latency, throughput, or usability scores. A reproducible follow-up should preserve command outputs, dependency versions, screenshots of key workflows and results of the same acceptance checklist on a fresh checkout.

C. Evaluation rubric

The reviewers constructed the following qualitative rubric before drawing the comparative conclusion. Each criterion is evaluated on documented implementation, reported verification and independent observation. A criterion is considered architecturally strong when the mechanism is explicit and its failure behavior is defined; it is experimentally confirmed only when the relevant test has been independently reproduced. This prevents a broad feature list from being mistaken for a validated security guarantee.

Table I. Reviewers' Qualitative Rubric And The Evidence Required For Each Claim.

Criterion	What the review checks	Finding and evidence boundary
Security architecture	Bound collector identity, separated roles, protected ingestion, persistence	Strong documented controls; local startup observed, security adversarial tests not independently rerun
Baseline governance	Manual approval, host/encoder separation, cold-start abstention	Explicit in implementation and reported tests; no labelled baseline-quality study
Evidence integrity	Canonical event bundle, SHA-256, Ed25519, independent verifier	Strong verifiability design; tamper checks reported, not independently reproduced here
Cloud safeguards	Dry-run default, target restrictions, recovery ledger, readback	Strong defensive design and mocked tests; no live AWS containment/recovery observed

D. Examined implementation and reported environment

According to the supplied verification record, the backend test environment used Python 3.13.5, FastAPI 0.128.2, Pydantic 2.13.4, Uvicorn 0.48.0, HTTPX 0.28.1, Cryptography 46.0.4 and pytest 9.0.2. SQLite provides event and baseline persistence; the React/TypeScript dashboard uses REST and authenticated SSE. Optional AWS services include EC2, Lambda, S3, DynamoDB and KMS, with Terraform modules for provisioning. Dockerfiles, Compose and a CI workflow are supplied, but their presence alone does not establish successful deployment.

The test fixture store is isolated from the application's normal runtime state. Startup does not create demonstration events, an approved baseline, or fabricated successful containment. An empty dashboard before telemetry ingestion is therefore an expected, honest application state.

E. Reported validation suite

The supplied verification report lists 78 Python tests distributed across collector, backend, cloud, detector, HTTP integration, utility scripts and security boundaries. One reported run completed in 4.18 seconds. The report also

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

records ten compiled frontend-core checks for SSE parsing, state replacement, ordering, bounded retention and counters; 29 TypeScript/TSX modules passed syntax transpilation. The transpilation check is not a full TypeScript typecheck, React render test, or production build. These test counts have not been independently reproduced by the three review authors.

Test area	Cases	Principal behavior exercised
Collector	13	Cursor/outbox, retries, rotation, identity
Backend	22	Correlation, persistence, baselines, evidence
Cloud response	14	Dry-run, target checks, preserve/readback (mocked)
Detection	7	Decoding, partial input, failure and vector binding
Local HTTP integration	1	Collector, API, signing, SSE replay/reset
Scripts and recovery	4	Independent verification and restore safeguards
Security	17	Roles, JWT/API keys, limits, origins
Total	78	Isolated validation suite

Table Ii. Distribution Of Python Test Cases Reported In The Supplied Verification Record.

F. Acceptance criteria not yet exercised

The reviewers' local run confirms that the backend and dashboard were usable in their development setup, but it did not include a real auditd host/fleet, labelled malicious/benign corpus, live MiniLM/Pinecone service, AWS staging containment-and-recovery drill, Terraform init/validate/plan, Docker runtime verification, or scale testing. The original delivery report also lacked the dependencies needed for a complete React typecheck, Vitest run and Vite production build. A separate Windows pytest attempt encountered platform-specific environment-length and open-file rotation issues; the final post-correction Windows suite result is not documented here. These limits do not negate successful local application startup, but they prevent treating it as proof of all deployment paths.

VI. REPORTED RESULTS AND CRITICAL ASSESSMENT

A. Outcomes in the supplied verification record

The supplied verification record reports 78 of 78 Python cases passing. Its local HTTP integration exercised movement of audit-format records from the collector through authenticated ingestion to persistent events, signed evidence and resumable SSE messages. Reported detector tests checked that an untrained baseline does not silently produce a score; evidence-tampering tests checked that modified content fails verification. Security tests covered role boundaries and identity binding, while cloud-response tests checked dry-run isolation and preserve-before-mutate ordering with mocked AWS clients. These observations are reported here with their original experimental limits.

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

The supplied report also records 10 passing frontend-core checks, 29 TypeScript/TSX modules transpiled without syntax errors and 64 local imports/exports resolved in source inspection. These results support implementation consistency but do not prove that every dashboard interaction was exercised in a browser. The reviewers ran the application locally but did not independently repeat and document the complete test suite.

Property	Recorded observation	Valid interpretation
Python validation	78/78 tests passed	Tested local functions and integration paths behaved as specified
Frontend core	10/10 checks passed	Streaming/state helpers passed targeted checks
TS/TSX syntax	29 modules transpiled	No syntax errors in checked modules; not full type safety
Local TCP pipeline	One integration test passed	Collector-to-API-to-evidence-to-SSE path exercised with audit-format fixture
AWS response	14 mocked cloud tests passed	Control-flow safeguards checked; no live containment demonstrated
Detection benchmark	Not performed	No measured precision, recall, F1, or false-positive rate

TABLE III. Reported Validation Results And Their Interpretation.

B. Independent observations and code-review findings

The local application run provides a narrower but original observation: the reviewed repository could be configured and launched and the dashboard interface operated sufficiently for interactive inspection. That finding supports local functional usability, not verified real-host telemetry or security effectiveness. The source and documentation review also revealed several concrete boundaries that matter when interpreting the implementation.

Finding	Positive implication	Remaining verification need
Local backend/dashboard launched successfully	Core application is runnable for review	Archive a fresh-checkout smoke-test log and environment details
No automatic demo telemetry or seeded baseline	Empty states do not masquerade as measured detection	Confirm with a fresh-state ingestion and baseline-approval trace
unscored is distinct from benign	Cold-start uncertainty is visible	Measure abstention rate and operational cost on labelled workloads
Test suite is reported on Linux; Windows pytest exposed portability issues	Platform assumptions are identifiable	Publish the corrected cross-platform test results
Cloud response is guarded but mock-tested	Safety checks can be inspected without mutating AWS	Independently test quarantine and restoration in an isolated AWS account

TABLE IV. Independent Local Observations And Artifact-Review Findings.

These findings are not defects of equal severity. The Windows test issue concerns portability of selected test procedures, not evidence that the backend/dashboard failed to start. The absence of seeded telemetry is an intentional design choice. Conversely, the lack of live cloud evidence is a genuine validation gap: source code and mocked tests cannot prove network isolation or successful recovery.

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2026, 11(4s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ **Research Article**

C. What the tests support

The principal evidence in the supplied test record is not a detection score. It is a set of reported behaviors that, if reproduced, would reduce opportunities for unsupported security claims. An absent baseline remains absent; a detector failure is not reported as a clean event; a modified evidence bundle fails verification; a dry run does not mutate cloud infrastructure; and a response job cannot claim successful quarantine merely because it attempted an API call. These properties are narrower than full system assurance, but they are relevant to the operational decision path the architecture was designed to protect.

The reported 4.18-second test-suite duration should not be presented as ingestion latency or throughput. It reflects execution of short isolated tests on one development environment. No confidence interval, event-per-second measurement, or attack-classification estimate follows from it.

D. Open performance and effectiveness questions

No labelled malicious/benign dataset was evaluated. Thus, precision, recall, F1, ROC-AUC, false-positive rate and detection delay cannot be compared numerically with published models. Similarly, no experiment has quantified whether operator review improves baseline quality enough to offset its workload cost. The evidence supports a locally runnable, architecturally integrated prototype with documented software tests. Independent repetition of the security tests and detection benchmarking are still needed to evaluate effectiveness.

VII. COMPARATIVE DISCUSSION AND LIMITATIONS

A. Why the integrated workflow stands out

Classical system-call models, DeepLog, LogBERT, LAnoBERT and NeuralLog chiefly investigate how abnormal behavior can be represented and detected [2]–[6]. The reviewed ZeroTrust SecOps AI implementation instead connects a deliberately modest detector to evidence verification, approval-gated baseline membership, authenticated investigation and guarded response. Among the works examined, this combination makes it a particularly complete reference example of governed novelty handling. That assessment is limited to the selected operational criteria: it does not establish superior sensitivity, precision, or fewer false positives and the studies answer different experimental questions.

Approach	Principal research emphasis	Relationship to this work
System-call models [2]	Program-behavior deviations	Earlier host-level detection foundation
DeepLog [3]	Learned temporal log patterns	Stronger sequence modelling; different input and evaluation
LogBERT / LAnoBERT [4], [5]	Transformer-based log anomaly modelling	Semantic/sequence benchmarks not replicated here
NeuralLog [6]	Parser-free log representations	Addresses representation errors; not directly benchmarked
ZeroTrust SecOps AI	Reviewed normality, abstention, signed evidence, response guards	Broader operational control path; accuracy unmeasured

TABLE V. Comparative Research Emphasis; Not A Head-To-Head Accuracy Ranking.

A feature-level comparison clarifies why the project stands out in this particular review. The published detector papers are assessed for the topics they principally investigate, not for an assumed absence of security features in implementations outside those papers. ZeroTrust SecOps AI has the broadest documented integration of the four

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

operational controls selected here, while the cited neural models retain the stronger published evidence on detection benchmarks.

Review criterion	Focus in cited model studies [2]–[6]	Examined ZeroTrust SecOps AI implementation
Baseline governance	Not the principal reported contribution	Operator approval, reason and host/profile separation
Insufficient-evidence handling	Different model-specific scoring protocols	Explicit unscored outcome
Evidence provenance	Not central to cited classifier evaluations	Canonical SHA-256/Ed25519 verification
Response and recovery	Outside cited anomaly-model comparisons	Dry-run, target guards, preserved state (mock-tested)
Detection effectiveness	Published benchmark results	No labelled benchmark result yet

TABLE VI. Selected Operational Criteria; Comparison Of Research Scope, Not Proof That Other Systems Lack These Features.

A simpler hashing encoder reduces model-serving dependencies and produces repeatable vectors, but it may miss semantic relationships learned by Transformer representations. Operator review makes baseline changes auditable, but analysts can still approve poor examples and the review process can become a bottleneck. A fixed threshold simplifies interpretation yet may behave unevenly across host roles. Cryptographic evidence strengthens integrity checks after signing, but it does not establish the truth of a compromised source. These are not incidental shortcomings; they define the trade-offs of the design.

B. Internal, external and construct validity

Internal validity. Unit tests and mocks establish code behavior under chosen inputs. They do not cover every concurrency, network, cloud-policy, or filesystem failure. A real Linux host may behave differently from a temporary audit-format file. The corrected Windows-specific test variants also demonstrate that portability requires platform-specific validation.

External validity. A single-process SQLite prototype is not evidence of fleet-scale capacity. Multi-worker coordination, tenant isolation, distributed queues, key rotation and retention administration remain outside scope. A security-group attachment readback is a narrower condition than confirmed isolation of all traffic.

Construct validity. Passing 78 software tests is not an intrusion-detection accuracy result. The minimum baseline count of 20 and the default similarity threshold of 0.55 were not optimized on labelled attacks. ATT&CK context is descriptive, not evidence of successful technique detection. Any comparison that assigns ZeroTrust SecOps AI a superior F1 score without common datasets would be unsound.

C. Experiments required for a stronger claim

The next review iteration should first reproduce selected tests from the public checkout: collector retry and deduplication, role-bound ingestion, evidence tamper rejection and cloud dry-run behavior. Commands, versions, outcomes and any platform-specific failures should be archived so the independent findings can be audited. A subsequent study should pair a controlled Linux auditd corpus with representative legitimate change and labelled attack scenarios. A careful evaluation would compare feature hashing with MiniLM on the same host splits and compare reviewed static, reviewed adaptive and automatic baseline policies. Precision, recall, F1, false-positive rate, abstention rate, detection latency, analyst approval effort and resource use should be reported together. For cloud response, a disposable AWS staging experiment should verify actual attachment changes, recovery and limits on

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

collateral disruption. Only then could the work argue empirically that its governance choices improve operational outcomes.

VIII. CONCLUSION AND RECOMMENDATIONS

The independent review finds ZeroTrust SecOps AI distinctive because it preserves a distinction security operations cannot afford to lose: unusual is not synonymous with malicious. The implementation developed by Arya Brijesh Tiwari keeps that distinction visible as telemetry moves through collection, representation, review, evidence, investigation and possible response. The prototype binds collectors to source identities, correlates execution records, withholds scores when evidence or baselines are inadequate, signs original event snapshots and keeps cloud containment disabled unless deliberately configured. These choices make the route from a novelty finding to an operational decision more inspectable and recoverable.

The reviewers additionally ran the public repository locally and observed a working backend and dashboard. The supplied project verification report records 78 passing Python tests and successful targeted frontend-core and syntax checks; the complete suite has not been independently reproduced and archived by the review team. Together, these observations support local usability and the documented feasibility of the design, not attack-detection superiority or production cloud readiness. The most credible next contribution would be an experiment that measures whether reviewed baselines and explicit abstention improve security decisions under realistic workload change, while quantifying the additional analyst effort. That study should be accompanied by live auditd collection, semantic-encoder benchmarking, scale measurements and an independently observed AWS containment-and-restoration exercise.

Under the four published review criteria, ZeroTrust SecOps AI is the most comprehensively integrated detection-to-decision reference among the selected works examined, while recognizing that those studies primarily evaluated classifiers rather than complete operations platforms. This is a bounded architectural assessment, not an overall ranking or a detection-performance superiority result. It is best understood not as a replacement for the strongest anomaly classifiers, but as a framework in which a detector's output remains a claim to investigate until identity, evidence, policy and response state justify further action.

Software attribution and reviewer independence ZeroTrust SecOps AI was developed by Arya Brijesh Tiwari and is cited as the sole source-code artifact in [13]. The implementor, Arya Brijesh Tiwari, can be contacted at aryabrijeshtiwarii@gmail.com regarding the ZeroTrust SecOps AI reference implementation. The three manuscript authors are independent reviewers, not developers, co-developers, collaborators, or representatives of Tiwari. Their contribution is the local smoke test, review rubric, documented findings and literature-based assessment.

REFERENCES

[1] S. Rose, O. Borchert, S. Mitchell and S. Connelly, Zero Trust Architecture, NIST Special Publication 800-207, 2020. doi: 10.6028/NIST.SP.800-207.

[2] C. Warrender, S. Forrest and B. A. Pearlmutter, “Detecting intrusions using system calls: Alternative data models,” in Proc. IEEE Symp. Security and Privacy, 1999, pp. 133–145. doi: 10.1109/SECPRI.1999.766910.

[3] M. Du, F. Li, G. Zheng and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in Proc. ACM SIGSAC Conf. Computer and Communications Security, 2017, pp. 1285–1298.

[4] H. Guo, S. Yuan and X. Wu, “LogBERT: Log anomaly detection via BERT,” in Proc. Int. Joint Conf. Neural Networks, 2021, pp. 1–8. doi: 10.1109/IJCNN52387.2021.9534113.

[5] Y. Lee, J. Kim and P. Kang, “LAnoBERT: System log anomaly detection based on BERT masked language model,” Applied Soft Computing, vol. 146, Art. no. 110689, 2023. doi: 10.1016/j.asoc.2023.110689.

[6] V.-H. Le and H. Zhang, “Log-based anomaly detection without log parsing,” in Proc. 36th IEEE/ACM Int. Conf. Automated Software Engineering, 2021, pp. 492–504. doi: 10.1109/ASE51524.2021.9678773.

[7] M. Landauer, S. Onder, F. Skopik and M. Wurzenberger, “Deep learning for anomaly detection in log data: A survey,” Machine Learning with Applications, vol. 12, Art. no. 100470, 2023. doi: 10.1016/j.mlwa.2023.100470.

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

[8] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. EMNLP-IJCNLP, 2019, pp. 3982–3992.

[9] W. Wang et al., "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained Transformers," arXiv:2002.10957, 2020.

[10] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe and B.-Y. Yang, "High-speed high-security signatures," Journal of Cryptographic Engineering, vol. 2, pp. 77–89, 2012. doi: 10.1007/s13389-012-0027-1.

[11] MITRE ATT&CK, "Command and Scripting Interpreter (T1059)," [Online]. Available: https://attack.mitre.org/techniques/T1059/ . Accessed: Oct. 2026.

[12] National Institute of Standards and Technology, Incident Response Recommendations and Considerations for Cybersecurity Risk Management, NIST SP 800-61 Rev. 3, 2025.

[13] A. B. Tiwari, "ZeroTrust-SecOps-AI," reference implementation, GitHub repository. [Online]. Available: https://github.com/tiwari20122003/ZeroTrust-SecOps-AI . Accessed: Oct. 2026.

[14] K. Ayub and R. AlShawa, "Zero-Trust Architectures for SecOps in ServiceNow: Enabling Scalable Security in Enterprise Platforms," in Proc. 2025 Multimedia University Engineering Conference (MECON), Jul. 2025, pp. 1–5, IEEE.

[15] P. J. Worth Jr., C. Cunningham, and I. Cardei, "Optimization of Zero Trust Principles in Modern Security Operations: A Framework of Minimum Standing Trust," 2026.

[16] M. N. Hassooni, "Evaluation of Hybrid Cloud Computing Techniques to Enhance Data Security in Digital Enterprises," The USA Journals TAJET, vol. 8, no. 6, pp. 64–78, 2026.

[17] A. K. Kodeboyina, B. Chittanoori, and S. R. Kommisetti, "Agentic AI for Autonomous Zero-Day Threat Detection in Zero-Trust Enterprise Security Architectures," in Proc. 2026 International Conference on Secure Information Systems and Technologies (ICSIST), Aug. 2026, pp. 336–343, IEEE.

[18] V. Maksymov, "Architectural Principles and Operational Practices for Building Secure Digital Infrastructure in Cloud Environments," 2025.

[19] G. Prasadu, G. Karthick, and V. V. S. S. S. Balaram, "CLOUDPROTECT: Harnessing the Power of Cloud-Native Orchestration for SecOps Automation," Decision Making: Applications in Management and Engineering, vol. 7, no. 2, pp. 16–37, 2024.

[20] S. K. Upadhyay and S. Kumar, "Components and Utilities of Cybersecurity," Journal of Library and Information Communication Technology, vol. 15, no. 1, pp. 137–147, 2026.

[21] R. K. Kripa, "KubeSecOps: Continuous Security Assurance in Kubernetes-Based DevSecOps Pipelines for AI Workloads," in Proc. 2025 IEEE International Conference on Advances in Data-Driven Analytics and Intelligent Systems (ADACIS), Nov. 2025, pp. 1–6, IEEE.

Copyright © 2026 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.