VeriFlow Live Evidence Governed Adaptive Cyber Resilience for Dynamic Environments

Yuval Fradkin

Independent Researcher

Manuscript type: Conceptual Framework and Research Agenda

Abstract

Modern cyber environments change faster than conventional assessment cycles. A source-code commit, dependency update, cloud-policy edit, identity change, newly exposed endpoint, or runtime transition can create an exploitable path that was absent moments earlier. Existing practices provide valuable but often separately operated capabilities for continuous monitoring, vulnerability detection, attack-graph analysis, incident response, and automated remediation. The unresolved systems problem is how to combine anticipation and action without allowing a plausible machine-generated signal to become an accepted security fact, or a proposed repair to become an assumed remedy.

This paper presents *VeriFlow Live*, a conceptual framework for evidence-governed adaptive cyber resilience. The framework maintains a time-indexed model of code, configuration, identity, dependency, exposure, and runtime relations; detects security-relevant changes; generates hypotheses about newly forming vulnerabilities and attack paths; and subjects every consequential claim to a six-stage Proof Gate covering reachability, observable impact, reproducibility, production relevance, root cause, and remediation validation. Responses are constrained by explicit authority boundaries, proportionality, reversibility, and auditability. Learning is outcome-gated: only verified findings, refuted hypotheses, validated repairs, and measured side effects may alter durable operational knowledge.

The contribution is not a claim that monitoring, attack graphs, prediction, or automated response are individually new. It is an architectural synthesis in which prediction produces testable hypotheses, proof governs escalation and action, and remediation remains untrusted until the verified path is shown to be closed. We formalize the core objects and state transitions, define safety invariants, relate the framework to adjacent research, and propose falsifiable research questions and an evaluation protocol. No empirical performance claim is made; the paper establishes a rigorous design and research agenda for implementation and comparative validation.

Keywords: adaptive cyber resilience, attack paths, causal security model, continuous monitoring, cyber defense automation, evidence governance, predictive vulnerability analysis, remediation validation

1 Introduction

Cybersecurity systems commonly observe assets, enumerate weaknesses, correlate alerts, rank risk, and initiate response workflows. These functions are indispensable, yet their operational separation creates a recurring epistemic failure: a technically plausible signal can be treated as an established risk before exploitability is demonstrated, while a syntactically valid patch or policy change can be treated as a completed remedy before its effect on the relevant attack path is measured. In dynamic environments, both assumptions can become stale quickly.

Continuous monitoring, as defined by NIST, supports ongoing awareness of information security and organizational risk [1]. Attack graphs model how multiple conditions can compose into an attack [3, 4]. Current incident-response guidance integrates preparation, detection, response, and recovery with organization-wide cybersecurity risk management [6]. Autonomic-computing research describes systems that monitor, analyze, plan, and execute through a knowledge base [7]. More recent work explores LLM-supported autonomic operation and vulnerability remediation [21, 22, 23]. These traditions establish much of the foundation required for adaptive defense. They do not, by themselves, impose a single evidence contract across prediction, confirmation, response, repair, and learning.

VeriFlow Live is proposed as that contract. It treats the cyber environment as a changing system whose security meaning depends not only on the existence of a weakness but also on current relations among software, identities, privileges, network exposure, dependencies, and runtime behavior. A change first creates a *hypothesis*. The system must then collect sufficient evidence to establish that the relevant condition is reachable, produces a security consequence, can be reproduced, applies to the deployed context, has a sufficiently understood causal mechanism, and can be closed by a validated intervention.

The biological analogy used in the originating concept note is architectural rather than literal. A living organism is continuously sensed, modeled, defended, and adapted; treatment is selected according to evidence and assessed by outcome. Similarly, VeriFlow Live organizes cyber resilience as a closed loop: *sense, model, hypothesize, prove, respond, validate, learn.* The analogy does

not imply that software behaves biologically, nor that all risk can be predicted or repaired autonomously.

This paper makes four contributions:

1. It defines an evidence-governed architecture that joins change observation, cross-domain modeling, hypothesis generation, proof, bounded response, remediation validation, and outcome-gated learning.

2. It formalizes the Proof Gate as a sequence of evidence obligations rather than a scalar confidence score.

3. It separates predictive bug and attack-path discovery from security adjudication: prediction prioritizes experiments but cannot authorize a material conclusion or action.

4. It provides a falsifiable evaluation agenda, including baselines, datasets, metrics, ablations, safety tests, and explicit conditions under which the framework should be rejected or revised.

2 Problem Statement and Research Questions

Let an operational environment at time t be represented by a state S_t containing code, build artifacts, configurations, identities, privileges, dependencies, network relations, and runtime observations. An event Δ_t transforms the environment into S_{t+1}. The security question is not simply whether Δ_t matches a known pattern. It is whether the change creates, modifies, or closes a path from an attacker capability to a protected asset under the actual trust boundaries of S_{t+1}.

Three gaps motivate the framework.

Context gap. A weakness may be present but unreachable, mitigated, or irrelevant to the deployed configuration. Conversely, several low-salience changes across domains may jointly create a serious path.

Evidence gap. Detection and prioritization systems often produce scores, probabilities, or recommendations. These values are useful for triage but do not establish reachability, impact, or causal mechanism. Machine-learning systems also face distribution shift and the limits of closed-world assumptions [13].

Closure gap. A response may suppress an alert without removing the causal path, may fail in production, or may create a regression. A remedy therefore requires evidence after intervention, not merely approval before intervention.

The corresponding research questions are:

RQ1
Can a cross-domain, time-indexed model identify security-relevant relations created by change more accurately or earlier than inventory-centered and isolated-domain baselines?

RQ2
Does the Proof Gate reduce false security conclusions and unsafe remediation without imposing prohibitive time-to-decision?

RQ3
Can predictive models improve the order in which hypotheses are tested while remaining epistemically subordinate to reproducible evidence?

RQ4
Does post-remediation path validation reduce recurrence and residual exposure compared with workflows that close findings when a patch or configuration change is issued?

RQ5
Can outcome-gated learning improve future prioritization without amplifying unverified model assumptions or automation errors?

3 Related Work

3.1 Continuous Monitoring and Exposure Awareness

NIST SP 800-137 frames continuous monitoring as the maintenance of ongoing awareness to support risk decisions [1]. NIST SP 800-137A extends this concern to organizational assessment across systems [2]. These publications motivate persistent observation and governance, but they do not prescribe the proposed claim-level sequence from reachability through remediation validation.

The NIST Cybersecurity Framework 2.0 organizes cybersecurity outcomes around governance, identification, protection, detection, response, and recovery [19]. VeriFlow Live is compatible with these outcomes but operates at a different level: it governs the evidentiary state of an individual security claim as that claim moves from change detection to validated closure.

Vulnerability severity frameworks such as CVSS provide standardized characteristics for communicating severity [8]. Current FIRST guidance encourages consumers to use Threat and Environmental metrics to move beyond generalized assumptions toward context-sensitive prioritization [24]. Even enriched scoring, however, does not reproduce a vulnerability or prove that a particular attack path exists. VeriFlow Live therefore treats severity as one input to a claim, not as a substitute for environmental evidence.

3.2 Attack Graphs and Knowledge Representations

Attack-graph research demonstrates how individual preconditions and exploits may compose into multi-stage paths. Sheyner et al. formalized automated attack-graph generation and analysis [3]; MulVAL used logical rules and configuration facts for multi-host security analysis [4]. Subsequent work has surveyed attack-graph generation and scalability challenges [5]. MITRE ATT&CK supplies a structured knowledge base of adversary tactics and techniques [9], while D3FEND relates defensive techniques to offensive behavior through a knowledge graph [10].

VeriFlow Live builds on this lineage but assigns the model a broader role. The model is continuously revised

by heterogeneous change events; uncertain edges remain hypotheses; evidence provenance and freshness are first-class; and the model must represent both path creation and path closure.

3.3 Bug Prediction and Learning Based Detection

Software-defect prediction estimates defect-prone modules or changes. A systematic review by Hall et al. found that model performance depends strongly on methodological choices and data quality [11]. Defects4J provides reproducible real-world faults for controlled software-engineering research [12]. These approaches can prioritize inspection and testing, but a prediction that a component is defect-prone does not demonstrate a security vulnerability or an exploitable path.

VeriFlow Live uses prediction in a deliberately bounded manner. Statistical anomaly scores, static-analysis findings, large-language-model suggestions, and graph changes may create or rank hypotheses. They cannot satisfy the Proof Gate by themselves. This distinction also responds to the broader warning that machine-learning systems accumulate hidden dependencies and operational debt [14].

3.4 Autonomic Computing and Automated Response

The monitor-analyze-plan-execute knowledge loop in autonomic computing established a general architecture for adaptive management [7]. Recent research has investigated whether LLM-based multi-agent systems can advance autonomic service maintenance [21]. Security orchestration and current incident-response guidance similarly connect detection to containment and recovery [6].

LLM-supported remediation studies demonstrate both the practical value of collaborative assistance and the need to adjust machine roles according to task complexity [22]. PenHeal couples automated penetration testing with LLM-generated remediation selection [23]. These systems strengthen discovery or recommendation. VeriFlow Live addresses a different control question: what evidence must exist before a predicted risk, proposed response, or claimed repair is allowed to change operational state or durable knowledge?

3.5 Causal Reasoning and Cyber Resilience

Causal reasoning distinguishes observation from intervention and provides a vocabulary for asking whether an action changed an outcome [15]. VeriFlow Live does not assume that a complete causal model of a production environment is attainable. Instead, it uses a pragmatic causal standard: the hypothesized mechanism must be explicit enough to design a discriminating test, connect the triggering condition to the observed consequence, and verify that an intervention closes the same path. Cyber-resilience engineering provides the broader objective of anticipating, withstanding, recovering from, and adapting to adverse conditions [16].

3.6 Novelty Boundary

The proposed novelty is a system-level evidence contract, not any single analytic technique. Five elements must coexist: (i) a time-indexed cross-domain state model; (ii) prediction that creates hypotheses without promoting them to facts; (iii) a multi-stage Proof Gate whose states include unknown and stale; (iv) remediation treated as an intervention hypothesis requiring path-specific post-change validation; and (v) a validated-memory boundary that prevents unverified inference from becoming durable operational knowledge. Existing approaches may implement one or several of these elements. The research claim is that binding all five into one governed lifecycle yields a distinct and testable architecture.

Table 1 positions the proposal relative to adjacent traditions. The distinctions are architectural tendencies, not claims that every implementation in a category behaves identically. The comparison states the proposed design boundary and should be tested empirically rather than accepted as evidence of superiority.

4 Framework Overview

Figure 1 presents the core loop. Sensors collect change events and runtime evidence. A living model updates the relevant subgraph rather than rebuilding an undifferentiated inventory. A hypothesis engine identifies newly formed or materially changed paths. The Proof Gate determines whether a hypothesis may be promoted. A response governor limits the actions that may be proposed or executed. Validation compares the post-intervention state with the verified causal path. Only then may the learning store retain the outcome as operational knowledge.

4.1 Sense

The sensing layer ingests semantically typed events rather than treating all telemetry as interchangeable. Example sources include version-control changes, dependency manifests, build outputs, infrastructure-as-code plans, cloud audit events, identity and entitlement updates, service discovery, network exposure, policy changes, endpoint observations, traces, and security-control results. Each observation carries provenance, collection time, scope, and integrity metadata.

4.2 Model

The living model is a temporal, attributed multigraph

$$G_t = (V_t, E_t, A_t, \Pi_t), \tag{1}$$

where V_t contains entities, E_t contains typed relations, A_t contains attributes, and Π_t records evidence provenance and freshness. Nodes may include code units, packages, artifacts, services, interfaces, identities, privileges, data objects, controls, and assets. Edges may encode calls, deployment, trust, authorization, network reachability, data flow, dependency, or mitigation.

Table 1: Conceptual comparison with adjacent approaches

Approach	Primary object	Treatment of prediction	Evidence before material action	Remediation closure criterion	Durable learning rule
Continuous monitoring	Control and risk state	Optional prioritization	Policy dependent	Control reassessment	Program dependent
Vulnerability management	Finding and severity	Risk scoring or enrichment	Often triage centered	Patch or status completion	Finding history
Attack graph analysis	Preconditions and paths	Path enumeration or ranking	Model dependent	Recompute or remove condition	Model update
Incident response automation	Alert and playbook	Classification and correlation	Confidence or approval thresholds	Workflow completion and recovery checks	Case outcomes
Defect prediction	Component or change	Central output	Not normally an exploit proof	Outside primary scope	Retraining labels
LLM-assisted security operations	Task, alert, or remediation proposal	Generation or decision support	Human review or system-specific guardrail	Recommendation, execution, or task success	Interaction or task outcomes
VeriFlow Live	Time-indexed security claim	Hypothesis generation only	Six-stage Proof Gate	Demonstrated closure of the verified path plus regression checks	Only validated or explicitly refuted outcomes

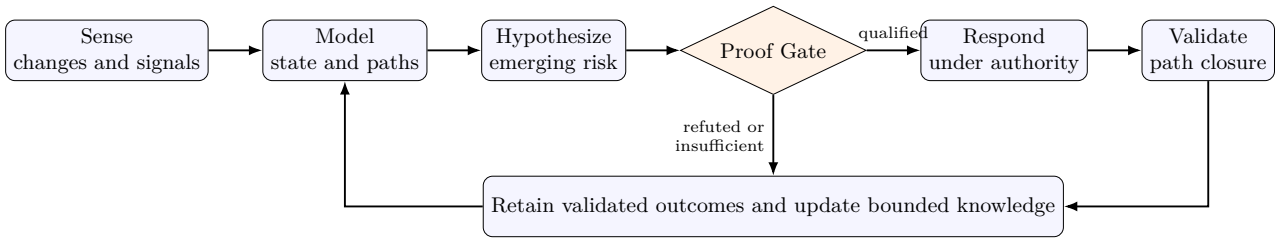

Figure 1: The VeriFlow Live evidence-governed resilience loop. Prediction creates hypotheses; it does not bypass proof.

The framework does not require all edges to be certain. Each relation has an epistemic state: *observed*, *derived*, *hypothesized*, *verified*, *refuted*, or *stale*. This prevents inferred relations from silently becoming facts.

4.3 Hypothesize

A change Δ_t induces a localized graph difference δG_t. Rules, program analysis, anomaly detection, learned models, or analyst input may propose a security claim

$$h = \langle \Delta_t, a, p, x, c, \tau, \rho \rangle, \tag{2}$$

where a is an attacker capability, p is a candidate path, x is the expected consequence, c is the deployed context, τ is the observation interval, and ρ is provenance. A hypothesis score may order experiments, but it cannot replace the evidence vector defined below.

4.4 Respond and Learn

Responses range from explanation and evidence collection to a prepared pull request, a tested policy change, or narrowly pre-authorized containment. Material or difficult-to-reverse actions remain subject to human authorization. Learning records both positive and negative outcomes, including false positives, failed reproductions, repair effectiveness, regressions, and side effects. Raw model speculation is excluded from the validated store.

5 The Proof Gate

The Proof Gate is an ordered set of six evidence obligations. For hypothesis h, define

$$\mathcal{E}(h) = \langle e_R, e_I, e_D, e_P, e_C, e_V \rangle, \tag{3}$$

where the components correspond to reachability, impact, deterministic reproduction, production relevance, causal understanding, and validation. Each component takes a value from

$$\{\mathsf{pass}, \mathsf{fail}, \mathsf{unknown}, \mathsf{stale}\}. \tag{4}$$

For a material security conclusion, the default qualification rule is conjunctive:

$$Q(h) = \bigwedge_{k \in \{R,I,D,P,C\}} [e_k = \mathsf{pass}]. \tag{5}$$

The validation element e_V applies to a proposed intervention m and is required before declaring the claim closed. Policy may permit an emergency containment action before full qualification when expected harm justifies it, but the action must be reversible, explicitly marked as precautionary, and must not transform the hypothesis into a confirmed finding.

5.1 Reachability

The system asks whether the relevant component, condition, identity, or interface can be reached in the current environment by the stated attacker. Evidence can include control-flow reachability, route exposure, effective

authorization, service configuration, package loading, or a controlled probe. Merely finding vulnerable code or a vulnerable version does not pass this stage.

5.2 Observable Impact

The triggered condition must produce a measurable security consequence under the threat model: confidentiality loss, integrity violation, authorization bypass, unsafe state transition, availability degradation, or another defined harm. A crash, warning, or anomalous output is evidence only to the extent that it establishes such a consequence.

5.3 Reproduction

The relevant behavior must be reproducible with recorded inputs, versions, configurations, and expected observations. Determinism need not mean identical timing; probabilistic failures can pass if their distribution and triggering conditions are measured and independently reproducible. Non-reproduction does not necessarily refute a claim, but it prevents unqualified promotion.

5.4 Production Relevance

Evidence must map to the deployed artifact, enabled feature, actual trust boundary, reachable interface, and effective control environment. Container reproduction alone, for example, cannot establish equivalence if the security property depends on a different kernel, identity provider, hardware boundary, or runtime policy.

5.5 Root Cause

The causal mechanism must be understood sufficiently to distinguish cause from symptom and to select an intervention. The standard is practical rather than metaphysical: the explanation must predict at least one discriminating observation and identify the conditions whose removal should prevent the consequence.

5.6 Validation

After intervention m, the original reproducer and relevant variants are rerun against S_{t+1}. Validation passes only if the verified path is closed, intended service behavior remains acceptable, no material substitute path is introduced within the tested scope, and evidence is linked to the deployed change. A closed ticket or merged patch is not itself validation.

6 Predicting Bugs and Emerging Attack Paths

The predictive component is a principal feature of the concept, but its scope must be precise. VeriFlow Live does not promise certain foreknowledge of unknown vulnerabilities. It anticipates where security-relevant conditions may be forming and converts that anticipation into testable hypotheses.

Table 2: Minimum evidence artifacts for the Proof Gate

Stage	Minimum artifact
Reachability	Path from attacker capability to triggering condition, with current configuration
Impact	Observable violation tied to a defined security property
Reproduction	Versioned procedure, inputs, controls, and expected result
Production context	Equivalence map for artifact, feature, configuration, and trust boundary
Root cause	Mechanism that predicts discriminating observations and a repair target
Validation	Pre/post evidence, path-closure test, and bounded regression/side-effect results

Three prediction levels are distinguished. *Component prediction* ranks modules or changes likely to contain defects. *Security-condition prediction* proposes a weakness class or violated invariant. *Path-formation prediction* identifies a combination of changes that may connect an attacker capability to an asset. The third level is the most distinctive because it combines evidence across code, identity, cloud, dependency, exposure, and runtime domains.

For example, a dependency upgrade may add a parser behavior, an identity change may grant a service account a new role, and a routing change may expose an endpoint. Each change alone may appear low risk. In the updated graph, however, their conjunction may create a path to a protected object. The framework generates h, identifies the weakest unverified edge, and selects an experiment with high expected information value.

Prediction is governed by four constraints:

1. **No semantic promotion by score.** A high probability ranks work; it does not assert vulnerability.

2. **Provenance preservation.** Every predicted edge retains the source model, input snapshot, and timestamp.

3. **Control comparison.** Experiments include negative or fixed controls where feasible, reducing confirmation bias.

4. **Outcome-gated update.** Training or retrieval memory distinguishes confirmed, refuted, unresolved, and stale labels.

This design separates the benefits of predictive search from the hazards of autonomous overconfidence. It also makes predictions scientifically assessable: a hypothesis has a timestamp, scope, expected observation, and later disposition.

7 Response Governance and Safety Invariants

Adaptive response creates risk because a correct diagnosis can still lead to an excessive intervention, and an incorrect diagnosis can propagate damage. VeriFlow Live therefore places the response engine under an authority policy $\mathcal{A}$ mapping evidence state, asset criticality, action class, reversibility, and expected harm to a permitted execution mode.

Four modes are proposed: *observe*, *recommend*, *prepare*, and *execute*. The default is not universal autonomy. Explanations and tests may be generated automatically; changes may be prepared in an isolated branch or sandbox; only narrow, pre-authorized, reversible containment may execute automatically. Material changes require review.

The framework maintains the following invariants:

1. **Evidence monotonicity.** New evidence may revise a conclusion, but uncertainty cannot be silently discarded.

2. **Claim-action separation.** Authority to test a claim does not imply authority to change production.

3. **Least authority.** An action uses the smallest scope and privilege sufficient for its objective, consistent with established protection principles [17].

4. **Reversibility preference.** When expected benefits are comparable, the more reversible intervention is preferred.

5. **Path-specific closure.** A claim cannot be closed merely because an alert disappears.

6. **Auditable provenance.** Every conclusion and action is traceable to evidence, policy, tool version, actor, and time.

7. **Validated-memory boundary.** Unverified inference remains outside durable validated knowledge.

8 Illustrative Scenarios

These scenarios demonstrate reasoning structure; they are not empirical results.

8.1 Source Change Creates an Input Validation Path

A commit removes a boundary check in a parser used by an externally reachable service. Static analysis produces a candidate integer-overflow condition. The living model links the changed function to the deployed artifact and endpoint. The hypothesis engine proposes a path from unauthenticated input to an out-of-bounds access. Reachability requires demonstrating the call path in the enabled feature; impact requires an observable violation; reproduction records the exact build and payload; production relevance maps the tested artifact to deployment; root-cause evidence contrasts vulnerable and fixed builds; validation reruns the exploit and boundary cases after the repair.

8.2 Identity and Cloud Changes Compose into Privilege Escalation

No software vulnerability is necessary. A policy edit grants a workload identity permission to impersonate another role, while a network change exposes the workload's administrative interface. The graph difference connects two previously separated conditions. Prediction identifies the new composite path. The Proof Gate tests effective permissions, endpoint reachability, observable access to a protected operation, and the actual cloud policy. A proposed restriction is validated by showing that the path is closed while required workload operations still succeed.

8.3 Remediation Masks a Symptom

An automated recommendation adds a signature that suppresses a malicious request. The original proof showed that multiple encodings reach the same vulnerable parser. Validation therefore includes semantic variants. If one variant remains reachable, $e_V = \mathsf{fail}$ even though the original alert no longer fires. The framework retains the failed remediation and its side effects, but does not mark the risk closed.

9 Evaluation Methodology

The framework requires empirical validation before claims of operational superiority can be made. A staged evaluation should separate detection benefit, evidentiary quality, response safety, and learning behavior.

9.1 Experimental Units and Corpora

Evaluation units should include time-ordered change sequences rather than only static snapshots. Candidate corpora include reproducible real-world defects such as Defects4J [12]; public vulnerability-fix pairs with pre-fix and post-fix builds; infrastructure-as-code scenarios; identity-policy mutations; and controlled microservice environments instrumented for network and runtime observation. Security cases must preserve disclosure and licensing constraints.

To test cross-domain composition, the benchmark should include cases in which no single event is sufficient but a sequence creates a path. Each case should provide ground truth for attacker preconditions, affected asset, expected consequence, causal change, and a validated repair. Negative cases should include suspicious but unreachable code, disabled features, mitigated versions, and changes that alter telemetry without altering risk.

9.2 Baselines and Ablations

At minimum, comparisons should include: (i) severity-only vulnerability prioritization; (ii) inventory plus exposure correlation; (iii) static attack-graph recomputation; (iv) prediction without a Proof Gate; and (v) a human-led workflow with the same underlying tools. Ablations should remove production relevance, root-cause analysis, post-repair validation, cross-domain edges, and outcome-gated learning one at a time.

9.3 Metrics

Detection metrics include precision, recall, time-to-hypothesis, and lead time before conventional label-

ing. Proof metrics include qualification precision, false-confirmation rate, evidence completeness, independent reproduction rate, and time-to-qualified-decision. Response metrics include path-closure rate, regression rate, residual-path rate, rollback frequency, unauthorized-action count, and mean blast radius. Learning metrics include calibration drift, repeated false-positive rate, and performance on temporally held-out cases.

Operational usefulness should not be reduced to a single score. A system that detects more candidates but materially increases false confirmation or unsafe action may be inferior. A multi-objective evaluation should therefore report a Pareto frontier across coverage, decision latency, analyst effort, and safety.

9.4 Hypotheses and Rejection Criteria

The following preregisterable hypotheses make the proposal falsifiable:

- H1: Cross-domain temporal modeling improves recall for composite attack paths over isolated-domain baselines without an unacceptable precision loss.
- H2: The full Proof Gate lowers false-confirmation rate relative to score-based escalation.
- H3: Post-intervention validation lowers residual-path rate relative to workflow-completion closure.
- H4: Outcome-gated learning reduces repeated false positives under temporal evaluation compared with learning from all model outputs.

The framework should be rejected or materially revised if its evidence stages do not improve decision correctness, if latency makes it unusable for the relevant threat class, if cross-domain modeling adds complexity without measurable path-discovery benefit, or if automated containment produces harm that cannot be bounded by the proposed authority policy.

10 Threats to Validity and Limitations

Incomplete observability. The model can only reason over available and trustworthy evidence. Shadow assets, encrypted traffic, missing identity context, or disabled telemetry can produce false negatives. Unknown must remain a first-class state.

Model error and staleness. Graph edges derived from configuration or telemetry may become stale. Freshness policies and revalidation are required; a previously verified path cannot be assumed indefinitely.

Evidence poisoning. An attacker may manipulate logs, tests, model inputs, or validation environments. Evidence integrity, source diversity, isolation, and independent controls are therefore part of the trusted computing base.

Reproduction limits. Some race conditions, hardware faults, distributed failures, and destructive impacts cannot be reproduced deterministically or safely. The gate must support bounded statistical evidence and explicit residual uncertainty rather than forcing a false binary conclusion.

Causal ambiguity. Complex systems may admit several sufficient causes. Root-cause evidence should be scoped to the verified path and should not claim a complete causal account of the environment.

Automation risk. Reversible containment can still disrupt service, and rollback may be incomplete. Human review, least authority, staged deployment, and fail-safe defaults remain necessary. The framework aligns with risk-management principles for trustworthy AI systems and with the Generative AI Profile's emphasis on governing risks specific to generative models, but it does not by itself guarantee trustworthy implementation [18, 20].

Generalization. A successful evaluation in controlled cloud or software environments would not establish effectiveness in operational technology, safety-critical systems, or highly adversarial environments. Each domain requires its own equivalence and authority policies.

11 Discussion

The core conceptual shift is from managing findings to governing claims. A finding is a tool output; a claim connects a condition, a current environment, an attacker, a consequence, and evidence. This distinction permits aggressive search and prediction without granting those mechanisms authority to define truth.

The same principle applies to remediation. A patch is an intervention hypothesis. Its acceptance as a remedy depends on whether it changes the outcome for the verified path while preserving required behavior. This closes a gap between software delivery and security assurance: merge success, deployment success, alert disappearance, and path closure are different events.

The framework's proposed novelty is therefore integrative and procedural rather than elemental. Continuous monitoring supplies ongoing state awareness; attack graphs supply compositional reasoning; predictive models prioritize investigation; incident response supplies operational action; causal testing supplies intervention logic; and resilience engineering supplies the adaptive objective. VeriFlow Live binds these capabilities through a common evidence lifecycle and validated-memory boundary.

This design also clarifies a productive role for large language models and agents. They may translate heterogeneous evidence, propose invariants, generate tests, explain paths, and prepare repairs. Their outputs remain hypotheses or artifacts subject to the same gate. The architecture consequently seeks to capture agentic speed without converting fluency or confidence into security authority.

12 Conclusion

VeriFlow Live is a conceptual framework for adaptive cyber resilience in environments where security-relevant

relations change continuously. It maintains a living, time-indexed model; anticipates emerging vulnerabilities and attack paths; converts predictions into testable hypotheses; requires explicit proof before consequential conclusions; governs response through bounded authority; validates that interventions close verified paths; and learns only from measured outcomes.

Publishing the framework before a complete implementation serves a specific scientific purpose. It exposes the architecture, evidence obligations, safety invariants, novelty boundary, and rejection criteria to scrutiny before empirical results can encourage post hoc changes to the claim. It also supplies a common specification from which independent implementations, benchmarks, and competing designs can be derived. The contribution at this stage is therefore not evidence of operational superiority, but a sufficiently precise and falsifiable research object whose assumptions and evaluation protocol can be challenged in advance. This sequencing is appropriate only because the paper states its empirical status explicitly and defines what subsequent evidence would support, limit, or refute the framework.

The framework does not claim that every risk can be predicted, that complete causal models are attainable, or that autonomous remediation is universally safe. Its narrower claim is that cyber defense can become more adaptive without weakening epistemic discipline when detection, prediction, verification, response, remediation, and learning are designed as one evidence-governed loop. The next step is implementation and comparative evaluation under the falsifiable protocol defined here.

Data Availability

No dataset was generated or analyzed for this conceptual paper. A future empirical study should publish benchmark definitions, evidence schemas, evaluation scripts, and non-sensitive results sufficient for replication.

Conflict of Interest

The author declares no conflict of interest.

Author Contribution

Yuval Fradkin conceived the VeriFlow Live framework and is the sole author of this manuscript.

References

[1] K. Dempsey, N. S. Chawla, A. Johnson, R. Johnston, A. C. Jones, A. Orebaugh, M. Scholl, and K. Stine, "Information Security Continuous Monitoring for Federal Information Systems and Organizations," NIST Special Publication 800-137, Sep. 2011. doi: 10.6028/NIST.SP.800-137.

[2] K. Dempsey, E. Takamura, P. Eavy, and G. Moore, "Assessing Information Security Continuous Monitoring Programs: Developing an ISCM Program Assessment," NIST Special Publication 800-137A, May 2020. doi: 10.6028/NIST.SP.800-137A.

[3] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proc. 2002 IEEE Symposium on Security and Privacy*, 2002, pp. 273–284. doi: 10.1109/SECPRI.2002.1004377.

[4] X. Ou, S. Govindavajhala, and A. W. Appel, "MulVAL: A logic-based network security analyzer," in *Proc. 14th USENIX Security Symposium*, 2005, pp. 113–128.

[5] K. Kaynar, "A taxonomy for attack graph generation and usage in network security," *Journal of Information Security and Applications*, vol. 29, pp. 27–56, 2016. doi: 10.1016/j.jisa.2016.02.001.

[6] A. Nelson, S. Rekhi, M. Souppaya, and K. Scarfone, "Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile," NIST Special Publication 800-61 Revision 3, Apr. 2025. doi: 10.6028/NIST.SP.800-61r3.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003. doi: 10.1109/MC.2003.1160055.

[8] FIRST, "Common Vulnerability Scoring System Version 4.0 Specification Document," Forum of Incident Response and Security Teams, Nov. 2023. [Online]. Available: `https://www.first.org/cvss/v4.0/specification-document`

[9] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "MITRE ATT&CK: Design and Philosophy," The MITRE Corporation, Technical Report MTR170202, July 2018.

[10] P. E. Kaloroumakis and M. J. Smith, "Toward a knowledge graph of cybersecurity countermeasures," The MITRE Corporation, 2021. [Online]. Available: `https://d3fend.mitre.org/resources/D3FEND.pdf`

[11] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012. doi: 10.1109/TSE.2011.103.

[12] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proc. 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 437–440. doi: 10.1145/2610384.2628055.

[13] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. 2010 IEEE Symposium on Security and Privacy*, 2010, pp. 305–316. doi: 10.1109/SP.2010.25.

[14] D. Sculley et al., "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 2503–2511.

[15] J. Pearl, *Causality: Models, Reasoning, and Inference*, 2nd ed. Cambridge, U.K.: Cambridge University Press, 2009.

[16] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, "Developing Cyber-Resilient Systems: A Systems Security Engineering Approach," NIST Special Publication 800-160 Volume 2 Revision 1, Dec. 2021. doi: 10.6028/NIST.SP.800-160v2r1.

[17] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, Sep. 1975. doi: 10.1109/PROC.1975.9939.

[18] E. Tabassi, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, Jan. 2023. doi: 10.6028/NIST.AI.100-1.

[19] C. Pascoe, S. Quinn, and K. Scarfone, "The NIST Cybersecurity Framework (CSF) 2.0," NIST Cybersecurity White Paper 29, Feb. 2024. doi: 10.6028/NIST.CSWP.29.

[20] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST AI 600-1, July 2024. doi: 10.6028/NIST.AI.600-1.

[21] Z. Zhang, F. Yang, X. Qin, J. Zhang, Q. Lin, G. Cheng, D. Zhang, S. Rajmohan, and Q. Zhang, "The vision of autonomic computing: Can LLMs make it a reality?" arXiv:2407.14402, 2024. doi: 10.48550/arXiv.2407.14402.

[22] X. Wang, Y. Tian, K. Huang, and B. Liang, "Practically implementing an LLM-supported collaborative vulnerability remediation process: A team-based approach," arXiv:2409.14058, 2024. doi: 10.48550/arXiv.2409.14058.

[23] J. Huang and Q. Zhu, "PenHeal: A two-stage LLM framework for automated pentesting and optimal remediation," arXiv:2407.17788, 2024. doi: 10.48550/arXiv.2407.17788.

[24] FIRST, "CVSS v4.0 Consumer Implementation Guide," Forum of Incident Response and Security Teams, 2026. [Online]. Available: `https://www.first.org/cvss/v4.0/implementation-guide`