

Provable Assurance for Agentic Systems

A Verification Framework for Systems That Reason and Act

Authors: Matthew Schwartz

Contributors: Cameron Smith

Date: April 2026

Version: 1.0

Executive Summary

Generative AI is changing software development by making code production abundant. Verification is now the constraint. In an agentic development model, the limiting factor is no longer whether teams can generate software quickly. The limiting factor is whether they can continuously prove that software is correct, secure, policy-compliant, and safe to operate.

This paper builds on a body of prior work exploring AI security, agentic system design, and the evolving threat landscape for reasoning systems. Earlier publications examined the security foundations of generative AI, the distinction between predictive and reasoning AI, threat models for AI agents, memory architectures for agentic systems, and strategic guidance for enterprise AI adoption. This paper extends that work into a deeper treatment of provable assurance: the formal, continuous, and evidence-driven verification model that agentic and neuro-symbolic systems require.

Security teams should move beyond an operating model centered primarily on penetration testing, periodic red teaming, and manual review. Those activities remain useful, but they are insufficient for systems that reason, plan, invoke tools, modify state, and act across permission and data boundaries. The next operating model must be built around provable assurance.

Provable assurance combines formal methods, neuro-symbolic reasoning, continuous security evaluation, evidence-driven policy enforcement, and high-fidelity system source-of-truth validation. The goal is not simply to detect defects after development. The goal is to make high-risk systems increasingly approvable by design.

Organizations that adopt agentic software delivery without building verification infrastructure will increase throughput faster than they increase trust. Organizations that build provable assurance capabilities will be able to scale software delivery while preserving confidence in correctness, security, and governance.

Customer Problem

Modern software teams are entering a new development regime. AI systems now generate application logic, infrastructure code, tests, documentation, and workflow automations at a pace that exceeds the capacity of traditional validation methods. This creates three immediate problems.

First, code volume increases faster than human review capacity. Security teams cannot manually assess every change in a high-throughput AI-assisted environment.

Second, reasoning systems introduce a different class of risk. Agentic and neuro-symbolic systems make assumptions, chain decisions, invoke external tools, access data, and traverse privilege

boundaries. These risks are not fully addressed by traditional application security activities that focus narrowly on implementation defects or known vulnerability patterns.

Third, current assurance models are episodic. Most organizations still rely on point-in-time testing, milestone reviews, and manually assembled evidence. That model does not scale when software is continuously generated, modified, and deployed.

The result is a widening gap between development speed and assurance confidence, with a growing number of downstream non-technical impacts including regulatory exposure, liability risk, erosion of customer trust, and organizational accountability gaps that traditional security models were never designed to address.

Core Thesis

The most important security and engineering investment for agentic software delivery is a provable assurance system.

Provable assurance is a layered operating model that combines:

- Formal reasoning about system properties
- Continuous verification of permissions, policies, and information flow
- Explicit testing of assumptions made by agentic and neuro-symbolic components
- High-confidence evidence collection across the delivery lifecycle
- Risk-tiered controls that align assurance depth to business impact

This is not a rebranding of existing AppSec, nor is it simply the next evolution of it. AppSec was built around a fundamentally different problem: finding and fixing implementation defects in software that humans wrote, reviewed, and understood. Its tools, processes, and mental models assume that the system under test follows deterministic logic authored by developers. Agentic and neuro-symbolic systems break those assumptions. They make decisions based on learned inference, chain reasoning across tool boundaries, operate under delegated authority, and modify state in ways that no human explicitly coded. The assurance discipline required for these systems must reason about intent, authority, assumptions, and information flow, not just code quality. That is a different discipline, not an incremental extension of the existing one.

Defining Provable Assurance

Provable assurance does not mean every aspect of an agentic system can be proven in the mathematical sense. That would be an unrealistic claim for most modern AI systems. It means decomposing trust into explicit assurance claims, applying the strongest feasible validation method to each claim, and continuously renewing the evidence that keeps those claims credible.

In practice, provable assurance combines several kinds of assurance:

- Proof-backed claims for selected properties that are precise enough to model mathematically
- Policy-checked claims for permissions, data handling, workflow constraints, and separation of duties
- Evidence-backed claims for provenance, configuration integrity, and release readiness
- Runtime-validated claims for invariants that must continue to hold in production
- Adversarially tested claims for behaviors that are better challenged through red teaming, evaluation, and counterexample discovery than through proof

Key definitions:

- **Agentic system:** a software system that can plan, select actions, invoke tools, and modify external state with limited human intervention
- **Neuro-symbolic system:** a system that combines learned inference with explicit symbolic representations such as rules, graphs, planners, or ontologies
- **Provable assurance:** an operating model in which each critical claim is linked to a validation method, evidence source, owner, and renewal trigger
- **Renewable approval:** a deployment or operating decision that remains valid only while its supporting evidence and assumptions remain current
- **Invariant:** a property that must remain true across relevant states, transitions, or actions for the system to stay within its approved safety or security envelope

Assurance Claims Matrix

The practical unit of provable assurance is not a generic control. It is an explicit claim supported by evidence. A mature assurance program should maintain a claim matrix or safety-case-like structure for every high-risk system.

Assurance claim	Preferred validation method	Representative evidence	Renewal trigger
Authorization boundaries are enforced for every tool and action	Formal modeling, policy-as-code tests, and negative authorization tests	Machine-checked policy results, access-control test results, signed policy bundles	Policy change, new role, new tool scope
Sensitive data cannot flow to unapproved sinks	Information-flow analysis plus runtime monitoring	Data-flow graphs, schema validations, DLP checks, invariant alerts	New integration, schema change, model or tool change
Tool use stays within approved contracts	Typed tool schemas, precondition checks, isolation, and allowlists	Tool inventory, attested tool manifests, invocation traces, sandbox policies	New tool, dependency drift, scope expansion
Deployed artifacts match reviewed artifacts	Provenance verification and configuration attestation	Signed build provenance, artifact signatures, deployment attestations	Rebuild, dependency update, environment drift
Agent behavior remains within the approved safety envelope	Adversarial evaluations, red teaming, and runtime anomaly detection	Evaluation results, replay logs, exception records, anomaly reports	Model update, prompt change, policy change, new task class

When a Claim Fails: Worked Failure Scenario

The matrix above describes the steady state. The real test of an assurance model is what happens when a claim breaks. The following scenario illustrates how the workflow responds to a failure in the “Sensitive data cannot flow to unapproved sinks” claim.

Detection. A runtime information-flow monitor detects that a customer-support agent has begun sending customer PII to a newly added summarization tool that was not in the approved tool inventory at the time of the last release approval. The monitor fires an invariant violation alert.

Triage. The alert is routed to the assurance control plane, which correlates it with the claim matrix. The violated claim is identified: “Sensitive data cannot flow to unapproved sinks.” The control plane retrieves the current evidence package for that claim and flags that the tool inventory has changed since the last approval, but the data-flow analysis was not re-run.

Containment. The system automatically restricts the agent’s access to the new tool pending review. The agent continues to operate with its previously approved tool set. No manual intervention is required for containment because the policy-as-code layer enforces the restriction based on the tool inventory mismatch.

Investigation. The security team uses the decision logs and tool invocation traces to reconstruct the reasoning chain. They find that a routine tool inventory update added the summarization tool with a broad data scope. The agent’s planner selected it because it matched the task requirements. No adversarial action occurred. The failure was a governance gap: the tool was added to the inventory without triggering a data-flow re-analysis.

Remediation. The team updates the tool onboarding process to require data-flow re-analysis for any tool that can receive data classified above a defined threshold. The summarization tool is re-evaluated, scoped to non-sensitive data classes, and re-added to the approved inventory with explicit data-handling constraints.

Re-approval. The evidence package is updated with the new data-flow analysis, the scoped tool contract, and the updated policy-as-code rules. The renewal trigger for this claim is reset. The agent regains access to the summarization tool under the new constraints.

Post-incident. The exception is recorded with full attribution: what changed, when, who approved the tool addition, why the data-flow analysis was not triggered, and what process change prevents recurrence. The exception record becomes part of the system’s assurance history and is available for audit.

This scenario demonstrates the key principle: a claim failure does not mean the assurance model failed. It means the model detected a gap, contained the impact, and drove a specific remediation. The alternative, discovering the data leak weeks later through a customer complaint or regulatory inquiry, is what happens without provable assurance infrastructure.

Why Traditional Security Models Are Not Enough

Penetration testing is **not** the same as red teaming. Red teaming is not the same as assurance. These distinctions matter.

Penetration testing is generally focused on identifying exploitable weaknesses in a target implementation. It is useful for surfacing concrete defects, misconfigurations, and control gaps. However,

agent-based security testing solutions are increasingly surpassing human penetration testers in both precision and recall while operating at significantly faster pace. As these capabilities mature, the traditional model of periodic, human-driven penetration testing will likely shift toward continuous, agent-driven vulnerability discovery, further reinforcing the need for the provable assurance model described in this paper.

Red teaming is broader. It explores system assumptions, adversarial pathways, misuse cases, and operational failure modes. For agentic and neuro-symbolic systems, red teaming should focus less on generic threat enumeration and more on the assumptions the system makes about truth, tool behavior, authority, policy constraints, state consistency, and goal interpretation.

Neither activity, by itself, is sufficient for systems that must be trusted continuously.

Security for reasoning systems is fundamentally about reasoning over problems. It requires teams to answer questions such as:

- What can this system infer?
- What actions can it take?
- Under what permissions?
- What data can influence its decisions?
- Where can data flow once accessed?
- What assumptions must remain true for the system to be safe?
- What properties can be proven, monitored, or continuously re-validated?

If those questions cannot be answered with evidence, the system is not operating under a high-confidence assurance model.

Threat Landscape for Agentic Systems

Prior work on AI agent security examined threat models, attack surfaces, and security considerations across tools, memory, and orchestration layers. This section extends that analysis into a concrete attack taxonomy specific to agentic and neuro-symbolic systems operating in high-consequence environments.

Before defining an assurance strategy, it is important to understand the concrete attack patterns that agentic and neuro-symbolic systems face. These are not theoretical. They represent real adversarial techniques that exploit the unique characteristics of systems that reason, plan, and act.

Prompt injection remains one of the most immediate risks. Direct prompt injection manipulates the instructions given to a language model to override intended behavior. Indirect prompt injection embeds malicious instructions in data sources that the agent consumes during normal operation, such as documents, web pages, or tool responses. Both forms can cause an agent to take unauthorized actions while appearing to operate normally.

Tool poisoning targets the external tools and APIs that agents invoke. If an attacker can compromise a tool's response, inject malicious payloads into its output, or substitute a legitimate tool with a malicious one, the agent will incorporate tainted results into its reasoning and downstream actions. The agent has no inherent ability to distinguish a legitimate tool response from a compromised one.

Delegation chain hijacking exploits multi-agent architectures where one agent delegates tasks

to another. An attacker who can influence any link in the delegation chain can redirect actions, escalate privileges, or exfiltrate data through the trust relationships between agents.

Context window manipulation takes advantage of the finite context that language models can process. By flooding the context with irrelevant or misleading information, an attacker can displace critical instructions, safety constraints, or policy rules, effectively causing the agent to forget its guardrails.

Permission escalation through reasoning occurs when an agent uses its reasoning capabilities to find logical paths to higher privileges. Unlike traditional privilege escalation through software vulnerabilities, this happens through the agent's own decision-making process. The agent may reason that it needs elevated access to accomplish a goal and find a legitimate-looking path to obtain it.

Symbolic rule manipulation targets the rule engines, knowledge graphs, and constraint systems that govern agent behavior. Adversarial inputs crafted to trigger unintended rule interactions, exploit gaps in ontologies, or create contradictions in policy logic can cause the symbolic layer to authorize actions it was designed to prevent.

Knowledge base poisoning attacks the data stores that feed symbolic reasoning layers. If an attacker can insert false facts, modify relationships, or corrupt ontological structures in a knowledge base, every downstream reasoning chain that relies on that knowledge inherits the corruption.

Goal drift through ambiguity exploits vague, conflicting, or underspecified instructions. An agent operating under ambiguous goals may gradually shift its behavior in directions that serve an attacker's objectives while remaining technically consistent with its instructions.

Agent-to-agent trust exploitation targets the implicit trust relationships that form when multiple agents collaborate within a shared environment. Agents often share state, exchange intermediate results, and inherit authority from one another. A compromised or misconfigured agent can propagate tainted reasoning, corrupted data, or escalated permissions to every agent it interacts with. Unlike delegation chain hijacking, which targets a specific handoff, this vector affects the broader trust fabric: shared memory stores, implicit authority assumptions, conflicting goal states, and cross-agent data leakage. As multi-agent architectures scale, the trust surface grows combinatorially.

These attack patterns are not mutually exclusive. Sophisticated attacks will chain multiple techniques. A prompt injection might cause an agent to invoke a poisoned tool, whose output manipulates the symbolic reasoning layer, which then authorizes a privilege escalation. Assurance models must account for these compound attack paths, not just individual techniques in isolation.

Strategic Position

We recommend a forward-looking assurance strategy built on five positions.

1. Verification is the new bottleneck

AI has compressed the cost of generating software artifacts. It has not compressed the cost of proving that those artifacts are safe and correct. Verification now determines trusted delivery capacity.

2. Agentic systems require reasoning-centric security

Security for agentic systems is not limited to code defects. It includes decision logic, permission boundaries, delegation paths, policy adherence, tool invocation safety, and data-flow integrity.

3. Neuro-symbolic systems create assumption surfaces

Neuro-symbolic systems often combine learned behavior with symbolic structures, rules, graphs, planners, or constraint solvers. Their failure modes may arise not only from software bugs, but from invalid assumptions, stale facts, incomplete ontologies, unsafe rule composition, or mismatches between symbolic policy and learned inference.

4. Formal methods must be used selectively but deliberately

Formal methods will not apply uniformly across all software. They should be concentrated where correctness, safety, privilege management, information flow, or mission impact justify deeper assurance. This is where the principles associated with DO-333 style formal methods thinking become highly relevant: use mathematical rigor where failure is expensive and ambiguity is unacceptable.

5. Approval must be renewable, not permanent

Point-in-time approval is a fiction for systems whose models, tools, dependencies, policies, and operating context change continuously. Assurance must be tied to live system state. Approvals should expire automatically when material conditions change, and re-approval should be driven by updated evidence, not by calendar cycles or manual review queues.

Future State

The target state is an engineering and security model in which critical systems become increasingly approvable by design.

In that model:

- System architectures expose machine-readable policy, identity, dependency, and data-flow context
- Assurance evidence is generated continuously rather than assembled manually at review time
- Formal analysis is applied to the most critical security and correctness properties
- Neuro-symbolic systems are evaluated for invalid assumptions and reasoning drift
- Release decisions are backed by evidence packages, not verbal confidence
- Runtime behavior is monitored against expected invariants
- Exceptions are explicit, time-bounded, and attributable

The end state is not zero human involvement. The end state is that human review focuses on the truly ambiguous, high-risk, and policy-relevant issues rather than serving as a manual substitute for missing verification infrastructure.

A candid assessment of the current tooling landscape is necessary. No single product or platform delivers provable assurance for agentic systems today. Existing security tools were designed for a different problem: scanning static code, testing deployed applications, managing vulnerabilities

in known software components. They do not reason about agent decision logic, track information flow through multi-step reasoning chains, validate symbolic policy consistency, or continuously re-evaluate approval conditions as system state changes.

This means organizations cannot simply purchase their way to the target state. The tooling gap is real and spans several areas: formal verification tools are mature for hardware and embedded systems but lack integration with modern AI development workflows; policy-as-code engines exist but are not designed to model agentic authority delegation; runtime monitoring tools capture telemetry but do not correlate it with assurance claims or renewal triggers; and no commercial product currently provides end-to-end evidence packages that link formal analysis, runtime invariants, and release decisions into a single auditable artifact.

The implication is that early adopters will need to compose solutions from multiple tools, build custom integration layers, and invest in internal capability development. The organizations that treat this as a build-and-integrate challenge rather than a buy-and-deploy decision will reach the target state faster. As the market matures, commercial platforms will emerge to fill these gaps, but waiting for them is not a viable strategy for organizations already deploying agentic systems in high-consequence environments.

Formal Methods

Formal methods are not new. They have been used for decades in engineering domains where failure carries unacceptable consequences. In avionics, DO-178C and its formal methods supplement DO-333 provide a framework for using mathematical reasoning to verify flight-critical software. In automotive systems, ISO 26262 requires rigorous verification of safety-critical functions. Railway signaling relies on EN 50128 for software assurance in systems where errors can be fatal. Nuclear instrumentation and control systems follow IEC 61513. Medical device software is governed by IEC 62304. In security evaluation, Common Criteria (ISO/IEC 15408) uses formal modeling to verify protection profiles for high-assurance systems. Semiconductor design has relied on formal verification for decades to prove correctness of hardware logic before fabrication.

What is new is not the technique. What is new is the application domain. Agentic systems now exhibit the same complexity, autonomy, and consequence that justified formal methods in those fields. They make decisions, traverse authority boundaries, invoke tools, and modify state. The case for applying formal reasoning to these systems follows directly from the precedent set by safety-critical engineering.

Teams building high-consequence agentic systems should explicitly revisit formal methods. Traditional validation techniques do not always provide sufficient confidence for systems with complex reasoning paths, safety constraints, or sensitive authority boundaries.

The mindset associated with DO-333 is particularly useful because it frames formal methods as a way to strengthen assurance claims, not as an academic exercise. Other standards reinforce this thinking. IEC 61508 provides a general framework for functional safety that applies across industries. ISO 26262 demonstrates how formal techniques scale in high-volume, commercially driven engineering. Common Criteria shows how formal modeling supports security assurance at the highest evaluation levels. Together, these standards establish a clear pattern: when systems are critical enough, stronger assurance techniques, often including formal reasoning for selected properties, become hard to avoid.

In practice, that means applying mathematically grounded techniques to prove or strongly verify selected properties such as:

- Authorization invariants
- Separation-of-duty constraints
- Allowable information flows
- State transition correctness
- Tool invocation preconditions
- Policy completeness and consistency
- Safety envelope preservation

This does not mean every feature requires formal proof. It means organizations should identify the properties that matter most and use stronger assurance techniques where they materially reduce uncertainty.

Neuro-Symbolic Assurance

Earlier work explored the distinction between AI systems that predict and AI systems that reason, as well as the practical implications of machine learning techniques for security professionals. The neuro-symbolic paradigm sits at the intersection of these two threads. This section builds on that foundation to address the specific assurance challenges that arise when neural and symbolic components are combined in agentic architectures.

Neuro-symbolic AI refers to the integration of neural network approaches with symbolic AI methods. The neural side includes deep learning, large language models, reinforcement learning, and other statistical pattern recognition techniques that excel at handling ambiguity, unstructured data, and generalization from examples. The symbolic side includes knowledge graphs, logic programming, rule engines, ontologies, constraint solvers, and formal planners that provide structured reasoning, logical consistency, and explicit constraint enforcement.

The combination is deliberate. Neural systems are powerful but opaque. They can generalize across noisy inputs but struggle to guarantee logical consistency or explain their reasoning. Symbolic systems are precise and interpretable but brittle. They require well-defined inputs and complete rule sets. Neuro-symbolic architectures aim to get both: systems that can learn from data and reason over structure. In agentic contexts, this often takes the form of a language model or learned policy operating within a symbolic framework of rules, permissions, plans, or knowledge representations.

Neuro-symbolic systems deserve dedicated security treatment because they combine two different risk surfaces.

The neural component may generalize, infer, rank, classify, or extract patterns from ambiguous data. The symbolic component may impose constraints, represent facts, enforce rules, or structure reasoning paths. The combined system can appear more interpretable and controllable, but that does not eliminate risk. It shifts risk into the assumptions that connect the two layers.

Security review of neuro-symbolic systems should examine:

- How symbolic facts are created, updated, and trusted
- Whether policy rules are complete and conflict-free
- Whether the neural component can bias or distort symbolic decisions
- Whether symbolic constraints can be bypassed through representation gaps

- Whether reasoning outputs remain valid when upstream data is stale, poisoned, incomplete, or adversarially influenced
- Whether permission and information-flow guarantees survive multi-step reasoning chains

Traditional threat modeling is still useful, but it is incomplete unless it includes assumption analysis for the reasoning substrate itself.

Continuous Security Evaluation

Most assurance programs are snapshot-based. Agentic development requires continuous evaluation.

Continuous security evaluation means the organization maintains a high-fidelity, continuously updated source of truth for the properties that matter to trust decisions. This includes:

- Current code and dependency state
- Identity and permission mappings
- Tool access scopes
- Data lineage and information-flow boundaries
- Policy definitions and policy changes
- Runtime telemetry and invariant violations
- Release evidence and exception history

This source of truth should support continuous verification questions such as:

- Has the system's permission model changed in a way that invalidates previous approval assumptions?
- Can sensitive data now traverse new paths?
- Has a new tool introduced action capability beyond the approved boundary?
- Has a symbolic rule set drifted from the implemented policy model?
- Do current runtime behaviors still match expected invariants?

Without this continuous source of truth, approval becomes stale the moment the environment changes.

Supply Chain Risk for Agentic Tooling

Agents do not operate in isolation. They invoke tools, call APIs, consume libraries, and depend on external services. Each of these dependencies carries its own supply chain risk, and the agent inherits all of it.

When an agent calls a tool, it trusts not only the tool itself but the entire dependency graph behind it: the packages the tool imports, the APIs it calls, the data sources it queries, and the infrastructure it runs on. A compromised dependency anywhere in that chain can corrupt the agent's inputs, outputs, or reasoning without any visible indication of tampering.

This is distinct from traditional software supply chain risk because the agent actively selects and invokes tools at runtime based on its reasoning. The attack surface is dynamic. A tool that was safe yesterday may pull a compromised dependency today. An API endpoint that returned trustworthy data last week may now be serving poisoned responses.

Continuous security evaluation for agentic systems must therefore extend beyond the agent's own code and configuration to include:

- Dependency graphs for all tools the agent can invoke, with continuous integrity verification
- Version pinning and reproducibility for tool dependencies
- Runtime validation of tool responses against expected schemas and value ranges
- Provenance tracking for data that enters the agent's reasoning through tool invocations
- Monitoring for unexpected changes in tool behavior, latency, or response patterns
- Isolation boundaries that limit the blast radius when a tool dependency is compromised

Tool inventories that track only access scopes are insufficient. They must also capture the transitive trust relationships that each tool introduces.

Observability and Explainability

Provable assurance requires the ability to observe and explain agent behavior. If you cannot trace why an agent made a decision, you cannot verify the properties that decision should satisfy.

Observability for agentic systems goes beyond traditional application monitoring. It requires capturing the reasoning process itself: what information the agent considered, what alternatives it evaluated, what tools it invoked and why, what constraints it applied, and how it arrived at its final action.

Explainability is the complement to observability. Where observability captures what happened, explainability answers why it happened in terms that humans can evaluate against policy, safety, and correctness requirements.

Together, observability and explainability require:

- Structured decision logs that capture each step in the agent's reasoning chain, including inputs, intermediate states, outputs, and data classification tags
- Tamper-evident tool invocation traces with protected payload capture or cryptographic references to secured payload storage, enabling replay and audit without indiscriminate data duplication
- Reasoning chain capture that records which facts, rules, and constraints influenced each decision
- Attribution tracking that links every action to the policy, permission, and delegation chain that authorized it
- Anomaly detection that flags reasoning patterns that deviate from expected behavior
- Replay capability that allows security teams to reconstruct and re-evaluate past decision paths
- Retention, redaction, secret-scrubbing, and access-control rules that keep assurance telemetry from becoming a shadow data lake of sensitive content

Without these capabilities, formal properties become theoretical claims rather than verifiable facts. An organization may define authorization invariants, but if it cannot observe the reasoning chain that led to an authorization decision, it cannot confirm the invariant held. Observability and explainability are not optional additions to a provable assurance model. They are foundational infrastructure. They must also be designed with minimization and containment in mind so that assurance data does not create a second-order privacy, insider-risk, or compliance problem.

How the Operating Model Works

Provable assurance becomes real when design-time controls, release evidence, and runtime validation are linked into a single approval loop.

1. Define the risk tier and the assurance claims for the system or change.
2. Encode permissions, tool contracts, identity boundaries, and data-flow constraints in machine-readable form.
3. Generate evidence during development and release, including tests, evaluations, provenance, policy results, and where justified, formal analysis outputs.
4. Gate release on an evidence package that maps each required claim to current evidence, open exceptions, and residual risk.
5. Monitor runtime behavior for invariant violations, policy drift, anomalous tool use, and unexpected changes in dependencies or operating context.
6. Trigger re-approval automatically when material conditions change, such as model swaps, policy changes, tool additions, dependency drift, or new data access paths.

The critical point is that approval is not a one-time event. It is a renewable decision backed by current system truth.

Worked Example: A Code-Maintenance Agent

Consider an internal agent that can read source code, open pull requests, run test suites, and file tickets, but cannot deploy to production or access production secrets.

Its assurance claims might look like this:

- The agent can only invoke approved repository and CI tools with least-privilege scopes
- Source code may be sent to approved analysis services, but secrets and production data may not leave the boundary
- The agent may propose infrastructure changes, but a human must approve changes that alter deployment permissions or production network paths
- Every pull request generated by the agent must be attributable to an initiating user, a policy set, a model version, and a reviewed artifact set

The evidence package behind those claims would include:

- Policy-as-code results showing that deployment and secret-management actions are denied to the agent identity
- Provenance and attestation records tying prompts, policies, code, and binaries to the reviewed release
- Decision logs and tool traces showing which repositories, files, and tools the agent actually touched
- Invariant monitors that alert if the agent attempts a prohibited tool call, accesses a new data class, or proposes a workflow outside its approved boundary

The renewal triggers would be explicit:

- Adding a new tool
- Expanding repository or CI permissions
- Changing the model, prompt stack, or symbolic policy layer
- Introducing a new external analysis service or data sink

This example illustrates the broader principle: provable assurance is not a claim that the agent will never fail. It is a claim that the critical boundaries are explicit, testable, attributable, and continuously re-validated.

Workstreams

Workstream 1: Assurance Architecture

Build a common assurance architecture that integrates code evidence, policy evidence, identity evidence, dependency evidence, and runtime evidence.

Key outputs:

- Assurance control plane
- Evidence-linked release decisions
- Machine-readable risk tiers
- Exception governance model

Workstream 2: Formal Property Identification

Identify the system properties that most justify formal reasoning or proof-oriented validation.

Examples:

- Authorization correctness
- Sensitive data flow constraints
- Tool-use restrictions
- State transition safety
- Mission-critical workflow integrity

Key outputs:

- Property catalog
- Formal-method candidate inventory
- Prioritized proof targets for high-risk systems

Workstream 3: Neuro-Symbolic Assumption Analysis

Create a repeatable method for identifying and testing assumptions embedded in neuro-symbolic systems.

Key outputs:

- Assumption taxonomy
- Review checklist for symbolic-neural coupling
- Adversarial evaluation plan focused on reasoning failures
- Drift detection for symbolic knowledge and policy layers

Workstream 4: Continuous Security Evaluation Platform

Develop the technical foundation for continuous security evaluation.

Key outputs:

- High-fidelity source-of-truth architecture
- Continuous policy and permission validation
- Information-flow monitoring
- Invariant testing and alerting
- Evidence dashboards for leadership and auditors

Workstream 5: Red Team Modernization

Refocus red teaming for reasoning systems.

This should include:

- Testing invalid assumptions, not just technical attack paths
- Challenging tool trust boundaries
- Exploring permission escalation through reasoning chains
- Probing policy contradictions and symbolic bypass paths
- Evaluating whether the system behaves safely under ambiguity, stale state, or manipulated context

Operating Model Changes

This strategy requires a shift in ownership.

Engineering teams remain responsible for correctness and operational readiness.

Security teams move from primarily manual defect-finding toward assurance design, policy engineering, validation strategy, and exception governance.

Platform teams build the paved road that makes high-confidence delivery easier than ad hoc delivery.

Leadership shifts success metrics away from raw speed and toward trusted throughput.

Governance, Accountability, and Regulatory Alignment

Agentic systems introduce a fundamental accountability question: when an autonomous agent causes harm, who is responsible?

Traditional software liability models assume that a human made the relevant decisions. Agentic systems break that assumption. An agent may chain together a sequence of tool invocations, data accesses, and state modifications that no single human explicitly authorized. The delegation path from human intent to agent action may span multiple reasoning steps, each introducing interpretation, assumption, and judgment that the human did not directly review.

Provable assurance must produce the evidence infrastructure that accountability requires. This includes:

- Decision attribution that traces every consequential action back through the delegation chain to the authorizing human, policy, or system
- Audit trails that survive multi-step reasoning chains and capture not just what the agent did, but why it believed it was authorized to do so

- Clear boundaries of autonomy that define what the agent can decide independently versus what requires human confirmation
- Exception records that document when the agent operated outside its expected boundaries, what triggered the exception, and how it was resolved

Regulatory and assurance frameworks are converging on these requirements, but they are not all the same type of instrument. The EU AI Act is binding law within its scope and imposes legal obligations for certain AI systems. The NIST AI Risk Management Framework and the NIST Generative AI Profile are voluntary guidance documents. ISO/IEC 42001 is a management-system standard that organizations may choose to certify against. These frameworks do not yet fully specify agentic architectures, but their direction is consistent: systems that make consequential decisions must be transparent, auditable, risk-managed, and subject to human oversight.

Organizations that build provable assurance infrastructure now will be better positioned to meet these regulatory requirements as they mature. Organizations that defer accountability infrastructure will face costly retrofits when regulatory enforcement begins.

The goal is not to solve every legal question about AI liability. The goal is to ensure that when accountability questions arise, the evidence exists to answer them.

What Teams Should Build Now

Teams should stop optimizing for an older mindset in which security is a downstream review function. In an agentic development world, the right build priorities are different.

The following maturity model provides a phased adoption path. Teams should establish each level before advancing to the next.

Level 1: Foundation

The first priority is establishing the basic infrastructure that all subsequent assurance capabilities depend on.

1. Define risk tiers for systems and changes, distinguishing between routine modifications and high-consequence changes that affect reasoning, permissions, or data flows.
2. Create machine-readable inventories for identities, permissions, tools, and data flows. These inventories must include dependency graphs for all tools the agent can invoke.
3. Establish basic evidence collection for code provenance, policy definitions, and runtime configurations.
4. Implement structured decision logging and tool invocation tracing for all agentic systems.

Level 2: Structured Assurance

With foundational infrastructure in place, teams can build systematic assurance processes.

5. Identify where formal methods can provide material assurance value and create a prioritized property catalog and claim matrix for high-risk systems.
6. Create a dedicated neuro-symbolic review model centered on assumption identification and validation.
7. Build evidence-linked release gating that uses continuous evidence, signed provenance, and explicit renewal triggers rather than point-in-time process compliance.

8. Establish governance structures that define accountability boundaries, decision attribution, and exception handling for agentic systems.

Level 3: Continuous Provable Assurance

The mature state integrates all capabilities into a continuously operating assurance system.

9. Make approvals renewable and continuously revalidated rather than permanent, with automated re-evaluation when the environment changes.
10. Reframe red team exercises around reasoning failures, authority boundaries, compound attack paths, and assumption invalidation.
11. Deploy continuous invariant monitoring that validates formal properties against live system behavior.
12. Implement full reasoning chain replay and audit capability for all high-risk agent actions.

What Good Looks Like

A mature organization can answer the following questions quickly and with evidence:

- What critical properties have been proven, strongly verified, or continuously monitored?
- What assumptions does the agentic or neuro-symbolic system rely on?
- What tools can it call, with what scope, on whose authority?
- What sensitive data can influence decisions or be exposed by actions?
- What changed since the last approval decision?
- Which assurances remain valid, and which must be re-established?

If the organization cannot answer those questions, then it does not yet have a provable assurance model.

Metrics That Matter

Mature programs also track whether the assurance model is actually working. Useful metrics include:

- Percentage of high-risk systems with a current assurance claim matrix
- Percentage of privileged tool invocations covered by provenance, policy checks, and runtime tracing
- Mean time to re-establish approval after a material change
- Exception age and exception volume for high-risk systems
- Invariant violation rate per thousand agent actions
- Replay coverage for consequential decisions and actions

Limits of the Model

Provable assurance has real limits, and those limits should be acknowledged plainly.

- It does not prove the full internal behavior of large models or eliminate uncertainty in learned components.
- It does not remove dependence on trusted infrastructure such as identity systems, logging systems, policy engines, and artifact stores.

- It does not prevent harm caused by vague business objectives, poor governance, or unsafe incentive structures.
- It cannot make third-party tools, models, or data sources trustworthy without corresponding provenance, contractual, and technical controls.
- It creates its own obligations around telemetry minimization, retention, and access control.

Risks and Counterarguments

A common concern is that stronger assurance will slow teams down. That risk is real if controls are manual, fragmented, or poorly targeted. The mitigation is risk-tiered automation and deliberate investment in assurance infrastructure.

Another concern is that formal methods are too expensive or too academic. That can be true when applied indiscriminately. It is not true when applied selectively to high-value properties where failure is unacceptable.

A third concern is that red teaming and penetration testing already cover these issues. They do not. They contribute important evidence, but they do not replace continuous, reasoning-aware assurance.

Recommendation

Organizations building agentic and neuro-symbolic systems should adopt provable assurance as a strategic direction.

This means moving beyond a model centered on downstream security review and toward a model built on:

- Formal reasoning for selected critical properties
- Continuous validation of permissions and information flows
- Explicit testing of reasoning assumptions
- Evidence-linked release confidence
- Renewable approval grounded in current system truth

Code generation is no longer the scarce resource. Trust is.

The organizations that win in the next software era will not be the ones that generate the most code. They will be the ones that can prove selected critical properties and continuously demonstrate, at scale, that the systems they build remain within approved bounds.

References

Safety and Formal Methods Standards

- RTCA DO-178C, Software Considerations in Airborne Systems and Equipment Certification, 2011
- RTCA DO-333, Formal Methods Supplement to DO-178C and DO-278A, 2011
- IEC 61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems, 2010
- ISO 26262, Road Vehicles — Functional Safety, 2018

- EN 50128, Railway Applications — Communication, Signalling and Processing Systems — Software for Railway Control and Protection Systems, 2011
- IEC 61513, Nuclear Power Plants — Instrumentation and Control Important to Safety, 2011
- IEC 62304, Medical Device Software — Software Life Cycle Processes, 2006 (amended 2015)

Security Evaluation Standards

- ISO/IEC 15408 (Common Criteria), Information Technology — Security Techniques — Evaluation Criteria for IT Security, 2022
- NIST SP 800-53, Security and Privacy Controls for Information Systems and Organizations, Rev. 5, 2020
- NIST SP 800-218, Secure Software Development Framework (SSDF) Version 1.1, 2022
- NIST SP 800-218A, Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile, 2024

AI Governance, Assurance, and Risk Frameworks

- European Union Artificial Intelligence Act (EU AI Act), Regulation (EU) 2024/1689, 2024
- NIST AI 100-1, Artificial Intelligence Risk Management Framework (AI RMF 1.0), 2023
- NIST AI 600-1, Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, 2024
- ISO/IEC 42001, Information Technology — Artificial Intelligence — Management System, 2023
- MITRE, The AI Assurance Landscape (v1.0), 2025
- MITRE, Risk Discovery Protocol for AI Assurance (v1.0), 2025
- AI Security Institute, Safety cases at AISI, 2024
- AI Security Institute, How can safety cases be used to help with frontier AI safety?, 2025
- OWASP AI Security and Privacy Guide, 2024
- OWASP Top 10 for Large Language Model Applications, 2025
- OWASP Top 10 for Agentic Applications, 2025

Supply Chain, Provenance, and Policy References

- SLSA, Provenance and Verification Specifications
- In-toto, Specification and Attestation Framework, v1.0
- Sigstore, Artifact Signing and Transparency Log Documentation
- Cedar Policy Language Documentation
- Open Policy Agent (OPA) Documentation

Foundational References

- Clarke, E.M., Grumberg, O., and Peled, D.A., Model Checking, MIT Press, 2018
- Baier, C. and Katoen, J.-P., Principles of Model Checking, MIT Press, 2008
- Garcez, A.d., Lamb, L.C., and Gabbay, D.M., Neural-Symbolic Cognitive Reasoning, Springer, 2009
- Marcus, G., “The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence,” arXiv:2002.06177, 2020

Prior Work by the Author

The following publications are available in the [ArtificialDiaries](#) repository.

- Schwartz, M., “AI Agents Security,” 2025. [GitHub](#)
- Schwartz, M., “Memory for Agentic AI,” 2025. [GitHub](#)
- Schwartz, M., “Machine Learning for Security Professionals: Beyond the GenAI Hype,” 2025. [GitHub](#)
- Schwartz, M., “Securing Enterprise Cognition: A CISO’s White Paper for the Generative-AI Era,” 2025. [GitHub](#)
- Schwartz, M., “AI That Thinks vs. AI That Predicts: A Polar Navigation Test,” 2025. [LinkedIn](#)
- Schwartz, M., “Navigating the Security Landscape of Generative AI,” 2025. [GitHub](#)
- Schwartz, M., “Developers Now Own AI Output, Not Just Code,” 2025. [LinkedIn](#)