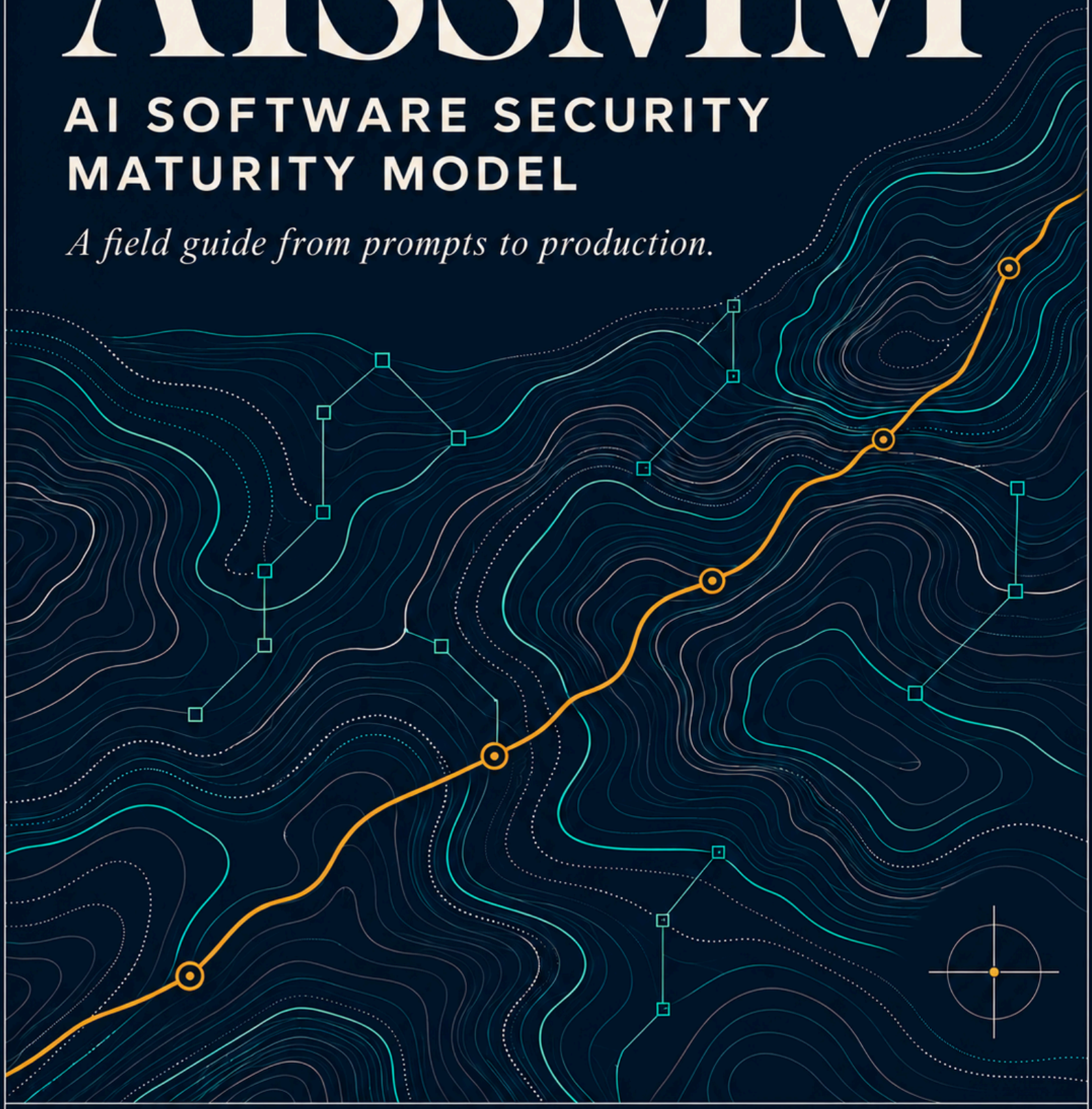
A CISO FIELD GUIDE

AISSMM

AI SOFTWARE SECURITY MATURITY MODEL

A field guide from prompts to production.



James Wickett

September 2026

Table of contents

Executive summary ... 3
Why AISSMM, why now ... 4
The four phases at a glance ... 5
P0 Visible ... 5
P1 Codified ... 6
P2 Operated ... 6
  Operating adversarial code review ... 6
  A gate the coding agent can clear ... 7
  Operating the program ... 7
P3 Assured ... 9
  Investigating existing exposure ... 9
  A maintained intelligence foundation ... 10
  Triage as organizational memory ... 10
  Assurance across the application portfolio ... 10
Assessing the current program ... 11
Governance and audit evidence ... 12
Reference architecture ... 13
Practical next steps ... 14
  Metrics for executive oversight ... 14
Implementation perspective: DryRun Security ... 15
The next AppSec era ... 15
  How the discipline evolves ... 16
  The application security role ... 16
  Research priorities for the next revision ... 16
  Conclusion ... 17
Sources ... 18

Executive summary

Our starting advice to an AppSec leader is direct: stop the bleeding. Every merge candidate, whether a pull request or merge request, needs adversarial code review before it enters the codebase. The reviewer must independently challenge the implementation's security assumptions and develop evidence for suspected failures. Coding agents can produce and repair the change, while the organization retains authority over acceptance as the volume of code grows.

The AI Software Security Maturity Model (AISSMM) guides teams toward AI code assurance: justified confidence that software developed with AI meets the organization's security requirements within a defined scope. Its four phases, P0 Visible through P3 Assured, describe an increasingly dependable program. Protection begins immediately, including while a team establishes its first inventory, with an accountable owner for unresolved material risk.

A useful gate gives the coding agent a practical way to clear it. Findings should explain the security failure and provide enough context for a fix that fits the application. The revised merge candidate then returns for independent verification. We recommend designing routine reviews to complete within five minutes, while defining a separate path for work that needs deeper investigation. This is an operating target, not a universal performance claim.

Adversarial code review must also reach the code already in production. A maintained intelligence foundation helps teams investigate package exposure and follow a discovered weakness across repositories. Reviewed triage decisions become organizational memory that informs subsequent analysis. P3 brings that knowledge together with enforceable verification, giving security leaders a defensible view of assurance across the application portfolio.

> **For a CISO,** the investment question is whether the program can prevent identified material risk from entering the codebase while reducing existing exposure. **For the head of AppSec,** the immediate task is to connect that objective to an effective merge gate, then build the intelligence needed to sustain it. AISSMM provides a practice grid and an improvement approach for evaluating that work.

Why AISSMM, why now

Our experience in AppSec has made us skeptical of programs that measure progress by how often they scan or how many tickets they create. The useful question is whether the organization can act on a security concern while the relevant code and its owner are still close to the work. AI-assisted development makes that question harder to postpone.

Stack Overflow's 2025 Developer Survey illustrates the gap between adoption and confidence. Among respondents to the usage question, 84% used or planned to use AI tools in development; in a separate accuracy question, 46% distrusted AI output and 33% trusted it. These are reported attitudes, not measurements of software security. They nevertheless reinforce the need to evaluate generated work rather than assume that adoption establishes reliability.

AISSMM grows from first-hand work helping AppSec teams adopt AI and act on its results. OWASP SAMM[1] provides a useful foundation for evidence-based practice assessment, while NIST's secure development and AI risk frameworks support broader governance. AISSMM applies that discipline to the daily operating problem of verifying AI-assisted changes. The model expresses a practitioner position that can be tested and improved through independent research.

Familiar vulnerability classes still matter, but reviewing application behavior requires more than identifying a suspicious pattern or an outdated package. A change can preserve the appearance of a correct implementation while violating an authorization boundary or a business rule. Secure base images and repository guidance remain useful; they cannot carry the entire assurance burden for the application the agent is writing.

> *"Application security teams are not asking for more findings; they are asking for better verification of which risks actually matter."*
>
> James Wickett, CEO, DryRun Security

[1]Related references include OWASP SAMM for security practice assessment, NIST SSDF for secure development, and NIST AI RMF for AI risk governance. DORA's AI Capabilities Model examines organizational conditions for adoption, while METR's 2026 follow-up illustrates the difficulty of measuring productivity gains. These sources support comparison and further research alongside AISSMM's practitioner foundations.

The four phases at a glance

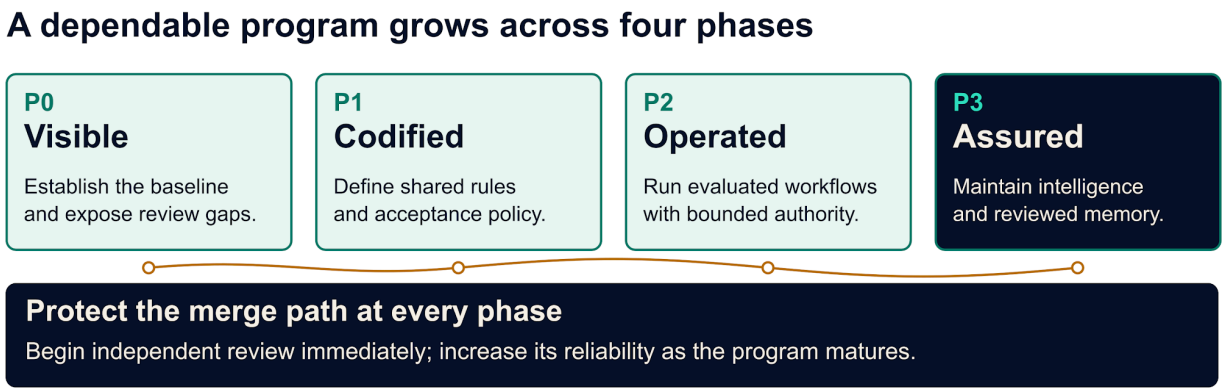


Figure 1. *Four phases of AI software security maturity.*

The phases describe observable capabilities within a defined application or portfolio. P0 makes the work visible, and P1 establishes shared requirements. P2 operates those requirements through evaluated workflows; P3 sustains them through maintained intelligence and organizational memory. Teams may advance at different rates across practices. The baseline recommendation remains the same throughout: evaluate every merge candidate, with review depth proportionate to risk.

P0 Visible

P0 Visible begins with an honest account of how AI contributes to software delivery. AppSec and engineering should identify which repositories use coding agents and who owns the resulting changes. They also need to understand the agents' access to sensitive code and credentials. The purpose of discovery is to help teams use these systems responsibly and expose gaps in the review path.

There is a sound reason to inspect the result independently. Chen and colleagues' SecureVibeBench study at ACL 2026 tested code agents on 105 C/C++ tasks from 41 open-source projects. The best-performing configuration produced solutions that were both functionally correct and secure on 23.8% of those tasks. This benchmark does not estimate the defect rate of an enterprise codebase; it demonstrates why functional completion alone is insufficient evidence of security.

Visibility should lead directly to protection. Select a representative repository, identify the merge authority, and put an independent review in the path of incoming changes. Where automation is incomplete, an explicit human review can bridge the gap. Record what remains uncovered and assign responsibility for it. A team should leave P0 with a credible baseline and a clear account of where material risk can still enter.

P1 Codified

At P1 Codified, teams make security expectations usable by both developers and their agents. Reviewed repository instructions should explain the application's security invariants and the tests needed to establish them. A named owner maintains those instructions as the application changes. The acceptance policy must also specify which findings prevent a merge and who may authorize an exception.

Skills and local harness configuration can improve the work an agent produces. Their presence, however, does not establish that a required check ran or that a failed result prevented acceptance. The distinction matters because instructions can become stale or may not load in a particular environment. Teams need evidence that shared requirements reach the intended workflow.

AGENTS.md is one convention for making repository guidance available to coding agents. Teams should verify support in the tools they actually deploy and keep personal preferences separate from organizational requirements. Centrally reviewed policy gives both developers and reviewers a stable basis for deciding whether a change is acceptable.

Adversarial code review investigates how an implementation could violate the application's security requirements. For example, a reviewer might trace whether a valid user can reach another tenant's records despite an apparent authorization check. The reviewer develops evidence for the suspected abuse path, with uncertainty explicit. An agent can perform this investigation when the surrounding system constrains its authority and evaluates its results. An enforceable acceptance boundary makes the review consequential.

Existing static application security testing (SAST) and software composition analysis remain useful sources of evidence. Contextual analysis should complement them where a decision depends on application behavior or business intent. A Codified program can explain how those methods support its requirements and how the merge control enforces the resulting decision.

P2 Operated

P2 Operated makes adversarial code review a dependable service across merge candidates and existing repositories. Reviews apply organizational policy and retain relevant application knowledge for later investigations. Each workflow has an accountable owner and bounded authority, with evaluation cases drawn from the applications it protects. The team measures decision quality within the delivery budget and defines recovery or exception paths for failed checks and unavailable reviewers.

Operating adversarial code review

An agent harness gives a model the tools and execution environment needed to perform a task. Orchestration governs how that task proceeds and what happens to its result. Ken Johnson's discussion of the distinction is useful because an impressive demonstration inside

a harness can conceal substantial operating work. The program still needs to route evidence and recover from failures without giving the model unrestricted control.

The acceptance boundary belongs to the surrounding system. A reviewer can recommend that a change proceed, but protected merge rules determine whether the required evidence is present and whether approval is authorized. This separation remains necessary when the authoring and reviewing agents use the same model family. Independence concerns decision authority and the quality of the challenge, not merely the number of model calls.

Evaluate the workflow under failure as well as normal operation. A malicious pull-request description should not be able to grant the reviewer new permissions, and a timeout should not silently become approval. These tests establish whether the control survives the conditions most likely to undermine it.

A gate the coding agent can clear

We recommend fast, actionable gates because coding agents can participate directly in clearing them. A remediation package should explain the violated requirement and show the evidence behind the finding. It should include the relevant code context and a verification condition, such as a reproducer or a test the repaired implementation must satisfy. The coding agent can then adapt the repair to the repository's conventions and submit the revised candidate for independent review.

A proposed fix is not a verified outcome. Automatically opening a replacement pull request can leave the application owner with another item to interpret, especially when the change ignores local design choices. A better workflow keeps remediation close to the authoring task and verifies the actual candidate that will merge. Human judgment remains available for ambiguous behavior and risk acceptance.

For routine merge-candidate review, design toward five minutes or less and measure performance across representative changes. Track slow reviews as well as typical ones, with explicit handling for unavailable analysis. Changes that need deeper investigation should enter a defined review path rather than receive an automatic pass. Speed matters because the control must fit the delivery process; its usefulness depends equally on the team's ability to act on the result.

Operating the program

Manifold Security's 2026 Azure DevOps MCP disclosure illustrates why agent authority deserves operational attention. Its proof of concept used instructions in a pull-request description to redirect a reviewing agent with broader permissions than the contributor. Treat retrieved project content as untrusted input, and constrain sensitive actions outside the model's discretion. A reviewer that can be persuaded to bypass its own control undermines the gate it was meant to support.

Building a review service requires ongoing evaluation and maintenance of the control boundary. A hosted service can transfer some of that operating work to a supplier, while internal or open-source implementations may offer a closer fit to local systems. Either

approach still needs an accountable program owner. Compare the full cost of producing a verified decision over a consistent scope, including the human work needed to resolve uncertain findings and maintain the service.

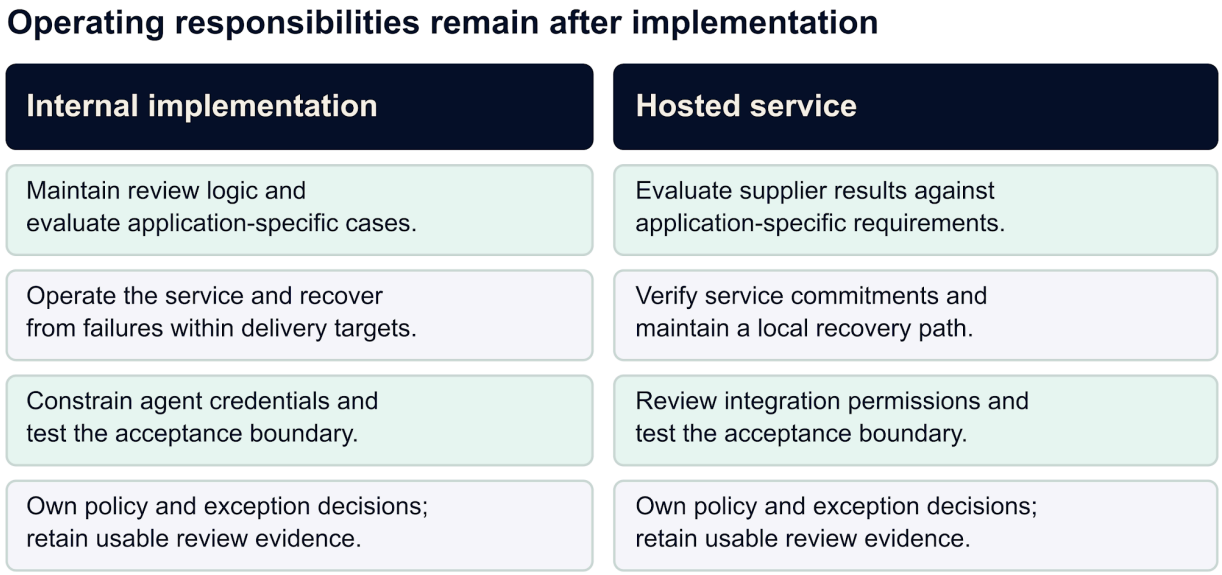


Figure 2. *Operating responsibilities when implementing a verification service. Responsibilities must have an owner regardless of the sourcing model.*

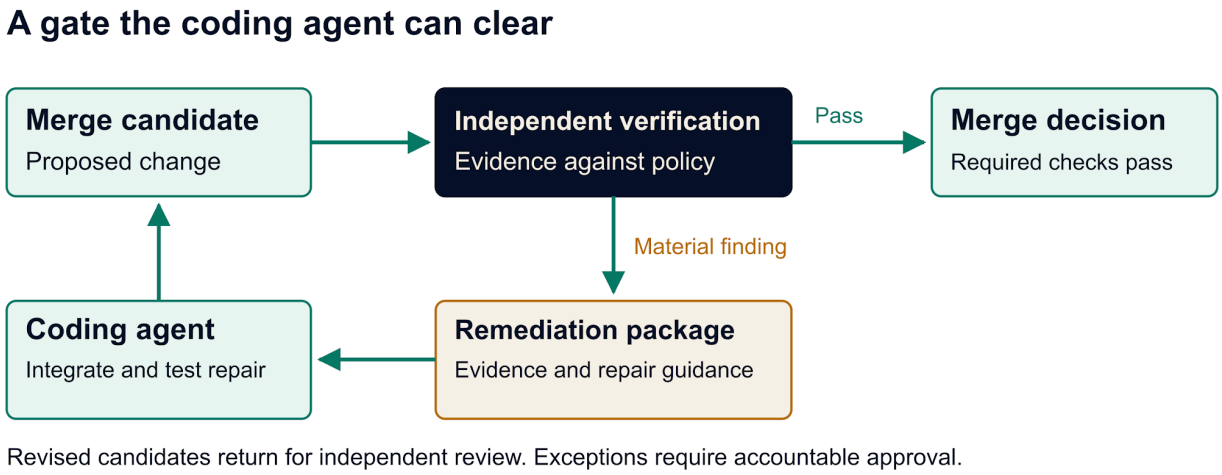


Figure 3. *The coding agent repairs a failed merge candidate using an evidence-backed remediation package. Independent verification governs the revised candidate.*

P3 Assured

P3 Assured makes AI code assurance an outcome the program can substantiate. Enforceable reviews use a maintained intelligence foundation, and reviewed triage corrections inform later decisions. The program can explain why a change satisfied policy and use the same knowledge to investigate existing exposure. Organizational memory makes this work cumulative across the repositories and applications in scope.

This approach responds to the verification constraint described by SRLabs:

> *"Security is not a pipeline of text transformations. It is a pipeline of verification... If you want a name for this, it is not an AI problem. It is a verification capacity problem."*
>
> SRLabs (Stephan Zeisberg), "AI-generated code, AI-generated findings," March 2026

Verification should produce evidence an engineer can challenge. A reproducer demonstrates an abuse path, while a reachability argument can establish whether vulnerable behavior is accessible. Retained reasoning lets another reviewer examine the decision and gives later investigations a starting point.

Investigating existing exposure

Extend adversarial code review into existing repositories, beginning with applications whose failure would cause the greatest harm. Investigations should trace suspected abuse paths in application context, with confirmed exposure assigned to an owner who can complete remediation. Retain the resulting knowledge so a finding in one repository can inform investigation elsewhere.

Package investigations require more than an inventory. Determine whether affected code is used in a relevant path and whether exploitation conditions are present, then find repositories that use the package similarly. The answer depends on application context as well as package identity.

The 2017 Equifax breach involved Apache Struts. GAO reported that the vulnerable dispute portal was missed during patching and by a subsequent scan. Log4Shell was a separate 2021 vulnerability in Apache Log4j. Both illustrate the need to connect an advisory to the applications that require action.

For the September 2025 Shai-Hulud campaign, CISA described malicious npm packages involved in credential theft and propagation. Our architectural conclusion is that runtime reachability alone cannot settle this exposure. Installation activity and possible credential compromise also require investigation, so the intelligence foundation must support those questions.

A maintained intelligence foundation

Repository understanding should accumulate as the program operates. A context graph can connect code behavior to dependencies and preserve the evidence used in earlier reviews. When a new advisory appears, the team can query that knowledge and target further analysis where the answer is incomplete. A bug-bounty report should similarly trigger a search for related behavior across repositories, followed by verification of plausible variants.

Code property graphs are one useful representation because they connect syntax to control-flow and data-flow relationships. The LLMxCPG research prototype uses graph-guided selection to focus analysis on relevant code. Its authors report code-size reductions of approximately 68% to 91% on the evaluated datasets. This supports investigation of the approach; local effectiveness still depends on the languages and applications being analyzed.

We recommend questioning the value of nightly or weekly whole-codebase scans that repeatedly reconstruct the same context and feed an unresolved ticket backlog. Maintained intelligence can support analysis of what changed and reuse earlier work when a new question arrives. The engineering requirement is freshness: code changes must invalidate affected conclusions, and a new attack may require evidence the system has never collected. Targeted reanalysis remains necessary when the stored context cannot support a decision.

Triage as organizational memory

When a reviewer corrects a finding, the organization has learned something about its software or the policy it intended to enforce. That reasoning should become part of the intelligence foundation so subsequent reviews can use it. Otherwise, the team repeatedly pays to explain the same context and risks teaching different reviewers conflicting lessons.

This requires more discipline than recording a dismissed alert. Retain the correction with its rationale and applicable scope, and establish when it must be reconsidered. A decision about one authorization path should not silently suppress a similar concern in an unrelated service. Reviewed feedback can improve future analysis without granting every dismissal the status of permanent policy. This gives the organization a durable record of accountable decisions that can inform later work.

At P3, the team can show that this feedback changes how the program operates. Sampled decisions should reveal which prior knowledge influenced a review and whether it remained applicable. Incidents and control failures should lead to updated evaluation cases or revised requirements. That continuing connection between evidence and practice is what makes the approach systemic.

Assurance across the application portfolio

Portfolio assurance makes coverage and uncertainty visible across applications. Apply shared criteria, using local evidence where deployment conditions differ. Dynamic testing and operational evidence address gaps in static knowledge, while architecture review examines unresolved assumptions and assigns owners for remaining risk.

Assessing the current program

Assess a named application or repository group against the practices below, using current evidence to identify the highest phase each practice demonstrates. The columns describe increasing reliability, rather than dates in an implementation plan. Start merge-candidate protection immediately and improve its operation alongside the other practices. The result is a planning profile; AISSMM has not been validated as a numerical scoring instrument.

Practice	P0 Visible	P1 Codified	P2 Operated	P3 Assured
Merge-candidate verification	Initial independent review; gaps recorded	Acceptance policy defines required checks	Evaluated gate controls each candidate	Gate effectiveness is continually tested
Agent remediation	Finding has an accountable owner	Repair evidence and expectations defined	Agent repairs; reviewer verifies	Reviewed outcomes improve later guidance
Existing code	Critical repositories identified	Exposure priorities and owners agreed	Verified investigations span repositories	New threats trigger focused variant analysis
Security intelligence	Code knowledge sources identified	Context requirements and ownership defined	Reusable repository context supports review	Freshness and invalidation govern reuse
Enforcement and ownership	Merge authority identified	Exceptions and approval rules defined	Protected controls enforce decisions	Sampled decisions establish accountability
Organizational memory	Triage reasoning is recorded	Corrections have rationale and scope	Reviewed feedback informs subsequent analysis	Retained decisions are tested for continued relevance

Table 1. *A practice profile across the four phases. Each cell describes evidence to seek, not a reason to defer a necessary control.*

Keep differences between practices visible. A team may operate a dependable review service while lacking a maintained view of its existing codebase. Another may have strong repository intelligence but weak exception handling. Choose the next investment according to the exposure the gap creates, and record the evidence that would demonstrate improvement.

Governance and audit evidence

AI-assisted delivery belongs within the organization's existing control environment. The program should make it possible to explain who authorized a change and why the available evidence was sufficient. The same account should remain intelligible when an agent performed much of the work.

The AICPA Trust Services Criteria used for SOC 2 address change management in CC8 and system operations in CC7. These areas help teams organize evidence of development and runtime controls. The applicable audit requirements depend on scope and the organization's stated commitments.

Deterministic controls establish the acceptance boundary, including required checks and authorized exceptions. Model-assisted analysis can investigate questions that need interpretation within that boundary. Record unresolved uncertainty and identify who is empowered to accept the associated risk.

ISO/IEC 27001:2022 Annex A controls A.8.25 through A.8.31 address secure development and related activities. Their applicability depends on the organization's risk assessment and Statement of Applicability. Use those commitments to determine the controls and retained evidence needed for AI-assisted work.

Governance reference points and evidence

CONTROL	AREA	POSSIBLE AISSMM EVIDENCE
SOC 2 CC8	Change management	Release reviews, timestamps and tests
SOC 2 CC7	System operations	Monitoring of AI-assisted deployments
ISO A.8.25	Secure development lifecycle	Codified policy and scoped workflows
ISO A.8.28	Secure coding	Reviewed AI-assisted coding rules
ISO A.8.29	Security testing	Evaluated tests and release verification
ISO A.8.31	Environment separation	Separate development, test and production

Figure 4. *Selected governance reference points and possible AISSMM evidence. This illustrative mapping does not establish audit conformity.*

Retain the evidence and policy behind each accepted candidate, including the rationale and approver for any exception. Protect these records through defined access and retention controls, and sample them to confirm that the recorded checks operated.

Reference architecture

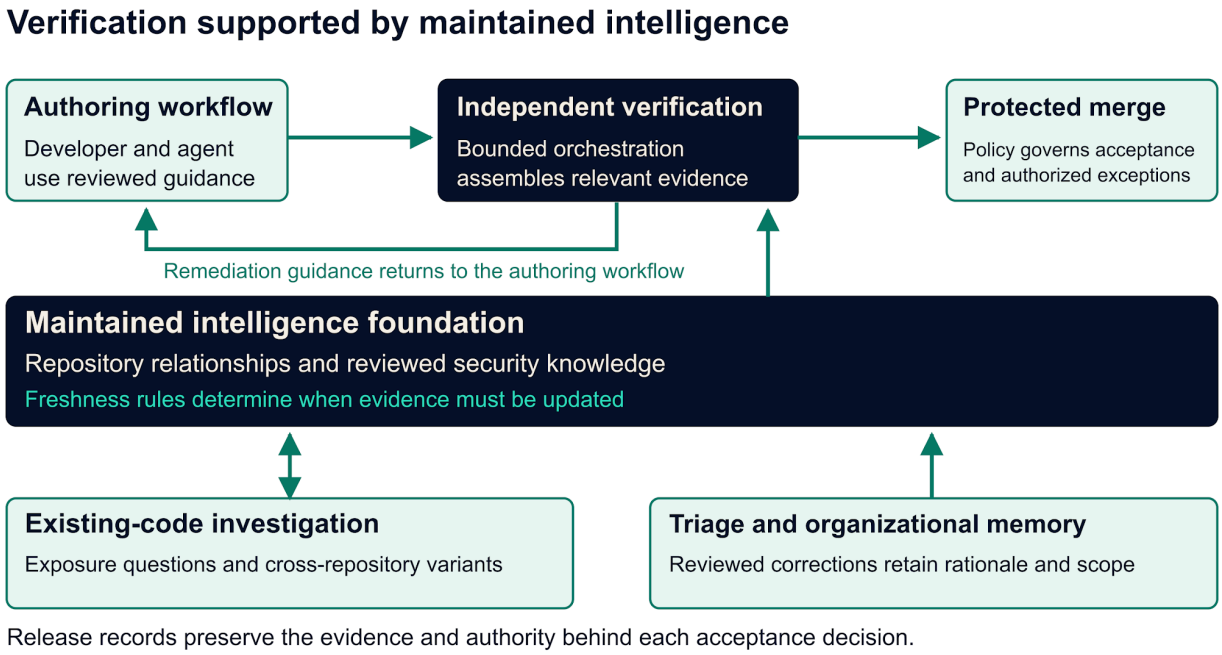


Figure 5. *Independent verification governs the merge decision. Maintained intelligence supports both change review and investigations across existing code.*

> The architecture connects the immediate merge decision to the longer-lived knowledge needed to support it.

Developers and coding agents work within reviewed repository guidance, then submit a merge candidate to independent verification. The orchestration service assembles relevant evidence while limiting the actions available to each agent. Protected merge rules enforce the acceptance policy; unresolved material findings return with a remediation package, and authorized exceptions remain attributable to the people who accept them.

The intelligence foundation supports that workflow and investigations of the existing codebase. It retains repository relationships and reviewed security knowledge so new analysis can build on prior work. Triage decisions feed back into this foundation with a defined scope, while changes in the code trigger updates to affected evidence. Release records preserve the basis of each decision even as current knowledge evolves.

These interfaces matter more than a particular product arrangement. An internal service or an external platform should be evaluated on whether it preserves the control boundary and produces useful evidence. The organization retains responsibility for its acceptance policy and for determining whether the knowledge supporting a decision is current.

Practical next steps

Begin by protecting incoming changes in a representative repository. Agree on what constitutes material risk, place an independent review in the merge path, and give the coding agent a usable remediation package. Confirm that failed checks prevent acceptance unless an authorized exception applies. A practical definition of the 'no net new risk' objective is that identified risk above the acceptance threshold cannot pass unresolved. It is not a claim that every vulnerability will be detected.

Once that path works, expand into the existing codebase and develop the intelligence foundation that will support it. Investigate a material exposure from end to end, including related use across repositories, and verify the resulting repairs. Use the exercise to identify which context can be retained and what must be refreshed. Review triage corrections with the application owners so organizational memory reflects deliberate decisions.

The phase profile helps choose where to strengthen this work. P0 and P1 teams need a credible baseline and shared acceptance requirements; P2 and P3 teams need evidence that operating controls remain dependable as conditions change. Expansion should follow demonstrated results and the ability to support additional applications, rather than an arbitrary calendar milestone.

Metrics for executive oversight

For executive oversight, begin with merge-gate coverage and verified remediation. Report the share of in-scope merge candidates that completed required checks, with bypasses shown separately. Measure how long confirmed material findings remain unresolved and whether repaired candidates pass independent verification. Keep the scope stable enough to distinguish improvement from a change in what was measured.

Delivery performance determines whether the control can be sustained. Compare typical review time with the slow end of the distribution, and examine how often developers must intervene to obtain a usable decision. Investigate incorrect findings alongside escaped defects. A high block rate by itself says little about whether the gate is protecting the application.

The intelligence foundation needs its own evidence of usefulness. Sample repository knowledge for freshness and test whether the team can answer a new exposure question across the intended scope. Track the time to a supported answer and the reanalysis required when existing evidence is insufficient. For triage feedback, examine whether corrections recur and whether retained decisions are being applied within their intended boundaries.

Calculate cost per verified change over a defined workflow, including attributable human review and service operation. Read that measure alongside coverage and decision quality, because a cheaper check can simply be doing less work. For existing-code investigations, assess cost against resolved exposure and reusable knowledge rather than the number of tickets generated.

Implementation perspective: DryRun Security

At DryRun Security, we approach this work through an adversarial code review platform. Reviews of merge candidates and existing repositories draw on a Code Security Knowledge Graph that preserves application context. Advanced verification examines package reachability and exploitability, while remediation guidance returns through our API and MCP integration to the coding agent doing the work. Reviewed triage feedback informs later analysis. This connected workflow illustrates how adversarial code review can support AI code assurance.

Veracode's 2026 Marketplace announcement identifies DryRun as its inaugural partner and describes combining established analysis with contextual AI reasoning. Its discussion emphasizes why the underlying knowledge matters:

> *"AI-led reasoning can surface vulnerabilities, but the quality of that reasoning is only as good as the underlying contextual understanding with which it's working."*
>
> Ajay Nigam, CPO, Veracode (Veracode newsroom, July 30, 2026)

The implementation question for a buyer is whether these capabilities work together in the buyer's own environment. Evaluate representative merge candidates and an existing-code investigation, then inspect the resulting evidence. A useful pilot should establish how a failed candidate is repaired and how a reviewed correction affects subsequent analysis. Those observations are more informative than review volume alone.

DryRun Security is the author's company, and this section reflects that first-party perspective. The phase criteria apply across implementations. Product claims and demonstrations should be evaluated against the same local evidence required of an internally built service.

The next AppSec era

AI-assisted development makes application security a more immediate engineering concern. Teams can produce code faster while still needing to understand whether that code respects the application's security assumptions. AppSec should help engineering meet that obligation through useful verification and a practical route to remediation. The work succeeds when developers can clear a credible gate and continue delivering.

This calls for a deliberate reconsideration of the advice to remove security gates from the development flow. Slow or unreliable controls deserved that criticism. A gate that supplies actionable evidence and can be cleared at the pace of AI-assisted work serves a different operating need. Our recommendation is to improve the gate's quality and response time, then measure whether it prevents material defects without creating avoidable work.

How the discipline evolves

The discipline must connect immediate verification to a longer view of the application.

Security leaders need to understand demand on the review service and whether engineering can act on its results. A growing backlog can expose a shortage of remediation capacity or poor evidence behind the findings. Increasing scan frequency will not resolve either problem by itself. Investment should address the constraint that prevents a supported finding from becoming a verified repair.

Independent judgment remains essential even when automation performs most of the work. Define the acceptance boundary and test whether it holds under adversarial conditions. Human review should focus on decisions whose ambiguity or consequence requires it, with responsibility explicit when remaining risk is accepted.

Business context gives technical evidence its significance. The same weakness can carry different consequences in an internal utility and a customer-facing service with sensitive data. AppSec works with application owners to make those distinctions available to reviewers and to keep them current as the business changes.

The application security role

Application security engineers increasingly maintain a system of verification and learning. They turn policy into checks that developers can understand, and they examine operational results to improve those checks. This role requires enough knowledge of the application to challenge a plausible answer and enough engineering judgment to make the resulting control dependable.

Organizational memory supports that work by preserving the reasoning that would otherwise remain with an individual reviewer. A well-maintained decision can help a later investigation, while an expired assumption can be identified and replaced. AppSec must govern that knowledge with the same care it applies to enforcement, because a convincing answer built on stale context can be worse than an explicit gap.

The profession's contribution is therefore visible in the organization's ability to make better decisions about its software. When the program can act on a new exposure quickly and explain the evidence behind its response, AI adoption gains a dependable basis for expansion.

Research priorities for the next revision

AISSMM is a practitioner model, and its phase criteria should be tested against evidence from operating programs. One research priority is whether independent assessors reach similar conclusions from the same evidence and whether improvements in a practice precede better security outcomes. Longitudinal studies should preserve application scope and account for changes in development volume before making causal claims.

OWASP SAMM offers a useful basis for comparing practice objectives and evidence expectations. A crosswalk should explain each correspondence and preserve differences in

scope, especially AISSMM's emphasis on merge-candidate verification and reusable code intelligence. DORA's AI Capabilities Model offers another comparison for organizational conditions that support adoption. Neither comparison establishes that AISSMM's phases have already been validated.

A complementary research priority concerns operating economics and developer experience. Compare internal and hosted implementations over the same review scope, including unsuccessful pilots. Evaluate whether remediation packages improve verified repair outcomes and whether maintained intelligence reduces the work needed to answer a new exposure question. Publishing the evaluation method and its limitations would make these findings useful beyond a single supplier.

Conclusion

AISSMM provides a path toward AI code assurance, with adversarial code review as a central operating practice. Protect the merge path now and give coding agents an independent gate they can clear. Extend that discipline into existing repositories, preserving application knowledge and reviewed decisions as organizational memory. The intended outcome is a defensible view of security across the application portfolio, grounded in current evidence and explicit about unresolved risk.

The founder perspective behind this paper is developed further in DryRun Security's essays Disruption, Not Deletion and Beyond Reachability. The sources below distinguish that perspective from external research and governance references.

Sources

OWASP. Software Assurance Maturity Model: model overview.

DORA. AI Capabilities Model (2025).

METR. We are Changing our Developer Productivity Experiment Design (2026).

NIST. SP 800-218: Secure Software Development Framework, version 1.1 (2022).

NIST. AI Risk Management Framework 1.0 (2023).

Stack Overflow. Developer Survey: AI (2025).

Chen et al. SecureVibeBench: Benchmarking Secure Vibe Coding of AI Agents via Reconstructing Vulnerability-Introducing Scenarios. ACL (2026).

SRLabs, Stephan Zeisberg. AI-generated code, AI-generated findings, and the verification bottleneck (2026).

Manifold Security. When Your AI Reviewer Works for the Attacker: Azure DevOps MCP disclosure (2026).

Lekssays et al. LLMxCPG: Context-Aware Vulnerability Detection Through Code Property Graph-Guided Large Language Models (2025).

AICPA. Trust Services Criteria and SOC resources.

ISO. ISO/IEC 27001:2022: Information security management systems.

AGENTS.md. Open format for coding-agent instructions.

DryRun Security, Ken Johnson. Your Agent Harness Is Not an Orchestration Layer (2026). Author-affiliated perspective.

Veracode. Veracode Marketplace launch announcement (2026). Vendor-reported partnership.

DryRun Security, James Wickett. Disruption, Not Deletion: How AI Is Rewiring Application Security (2026). Author-affiliated perspective.

DryRun Security. Beyond Reachability: The Exploitability Advantage in AppSec (2025). Author-affiliated perspective.

GAO. Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach. GAO-18-559 (2018), pp. 11 and 15.

CISA et al. Mitigating Log4Shell and Other Log4j-Related Vulnerabilities. AA21-356A (2021).

CISA. Widespread Supply Chain Compromise Impacting npm Ecosystem (September 2025).

SPONSORED BY

AISSMM

AI Software Security Maturity Model

V0.8

© 2026 DryRun Security.
This work may be shared with attribution under CC BY-NC-SA 4.0.