SACR

NOVEMBER 2025

AGENTIC REMEDIATION: THE NEW CONTROL LAYER FOR AI-GENERATED CODE

AQSA TAYLOR
HENRY HERNANDEZ

LEGIT SECURITY

Authors

- **Aqsa Taylor** Chief Research Officer, leading research division, CISO arm and AI product development at SACR. She is a published author of two cybersecurity books and numerous research articles on cloud security and SecOps. She brings a decade of experience as a Product Leader with a track record of building some of the well known security platforms such as Twistlock, Prisma Cloud Workload protection, Prisma Cloud Agentless scanning, Gutsy (Minimus), merging cybersecurity, AI, and cloud infrastructure. With deep experience across early-stage startups, she has led stealth to launch product efforts, integrated multi-million dollar acquisitions, and delivered features that drive real value for users. She is also a public speaker with writings recognized across several renown media outlets.
- **Henry Hernandez** is an expert on cloud security and identity. With 25 years at the intersection of technology, sales engineering, and cybersecurity, he has helped shape how enterprises think about cloud, SaaS, and identity protection. From leading roles at security vendors such as Citrix, Splunk, and Palo Alto Networks to advising startups on go-to-market and cloud-native strategy, his work bridges deep technical insight with real-world execution.

Table of Contents

Key Insights

AI now generates nearly half of enterprise code, creating significant security challenges. While productivity has increased, oversight has lagged. Developers often struggle to remediate flaws in AI-generated code they did not author, and prompting the model to retry can exacerbate issues. Recent studies confirm this trend (GitHub Octoverse, 2025; Stack Overflow Developer Survey, 2025).

A University of San Francisco study (2025) found that after five rounds of refinement, critical vulnerabilities increased by 37 percent. Although development speed improved, risk exposure also grew. Breaches involving AI-generated logic now cost between four and nine million dollars per incident, and unpatched flaws result in average compliance fines of half a million dollars per month (IBM Cost of a Data Breach, 2025; Verizon DBIR, 2025). CISOs must weigh time savings against financial risk.

Traditional application security tools were designed for human intent and context, both of which AI-generated code disrupts. A 2024 enterprise case study found that remediating AI-generated code took three times as long as remediating human-written code. Teams first had to determine the code's purpose before repairing it, making the challenge both technical and contextual.

Agentic remediation addresses these challenges by identifying issues, generating and testing fixes, and documenting actions. With structured validation, success rates now exceed ninety percent. AI is shifting from a risk factor to an essential component of defense.

This report examines this transition and highlights how select vendors are deploying agentic remediation at scale.

Actionable Summary

Security leaders should proactively address risks from AI-driven development before they escalate. The following steps provide a practical approach to preparing for and scaling agentic remediation within enterprise environments.

Discover AI-generated code early.

Use AI-BOM or PBOM scanning to identify where AI-assisted commits enter repositories and pipelines. Early visibility establishes accountability before incidents occur.

Measure the remediation gap.

Track the average time to remediate AI-generated code compared with human-written code. If remediation takes two or three times longer, this indicates a deeper structural issue in the workflow.

Run controlled pilots.

Start with semi-autonomous pull requests that generate fixes for human review. Begin with lower-risk systems to test accuracy and developer confidence before broader implementation.

Build validation into the process.

Require every AI-generated fix to pass static analysis, integration testing, and runtime fuzzing before merging. This approach prevents recurring errors and maintains trust in the process.

Expand carefully.

Increase automation only where risk is low and outcomes are measurable. Use pilot results to inform policy before full rollout.

Agentic remediation is now operational. It enables security teams to keep pace with AI-driven development without sacrificing assurance. The goal is not to replace human oversight but to restore balance between speed and security.

The following sections expand on these steps and profile the vendors now enabling these capabilities in production environments.

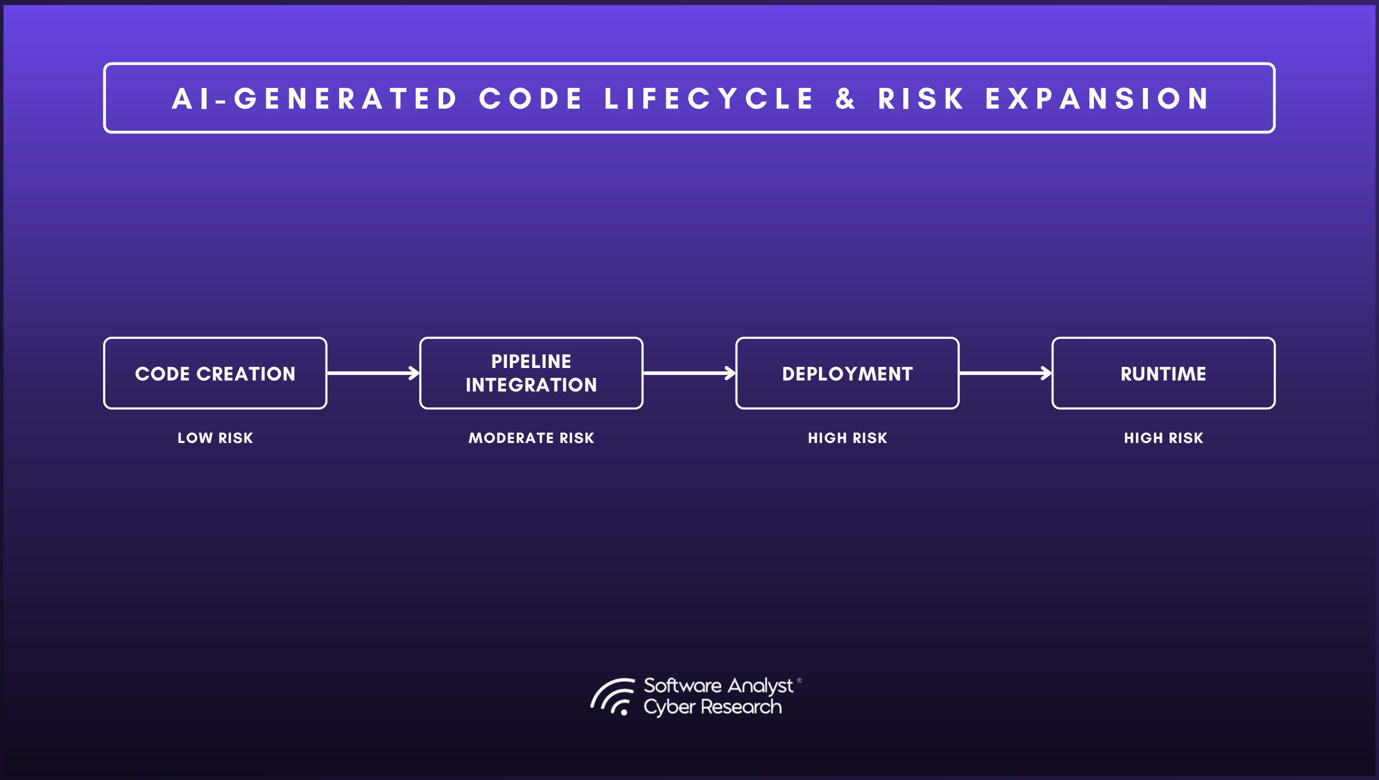
Introduction & Market Context

Problem Statement

AI now produces between thirty and fifty percent of enterprise code (GitHub Octoverse, 2025; Stack Overflow Developer Survey, 2025). Productivity has accelerated, but oversight has not kept pace. Incident alerts are increasing as AI-generated code exposes sensitive data through missing validation and excessive permissions. Security teams face a growing operational burden as developers move faster than traditional controls can keep pace.

This imbalance introduces a new class of risk. Developers benefit from AI's speed, but security teams face code they did not design and cannot easily explain. Traditional remediation workflows depend on author intent and contextual understanding. Those assumptions no longer hold. When a vulnerability appears in AI-generated logic, there is often no clear lineage or rationale behind it.

A study by the University of San Francisco (2025) found that after five refinement rounds, critical vulnerabilities increased by thirty-seven percent. The result is a widening remediation gap and rising costs. Enterprises must now secure code produced by systems that lack accountability, context, and explainability.



Market Evolution

AI-generated code has moved from experimentation to production. Most enterprise development pipelines now rely on AI assistants that continuously generate and modify code. These systems deliver functional results fast, but they also create architectural complexity that weakens traditional AppSec workflows.

Security tools built for static analysis and human authorship are now misaligned with AI-driven output. Scanners flag vulnerabilities, but developers hesitate to modify code they did not author, increasing mean time to remediation. The absence of authorial context means even minor fixes require time-consuming reverse engineering.

As this gap grows, the market is shifting toward platforms that can both create and correct code autonomously. Agentic remediation combines discovery, validation, and continuous feedback to manage AI-written software at scale. Vendors in this space are embedding these capabilities directly into developer workflows.

This marks the transition from detection-first security to proactive control. Security is no longer just about finding vulnerabilities; it is about ensuring that fixes are explainable, validated, and auditable.

The AI-Generated Code Challenge

AI introduces new patterns of vulnerability that traditional security tools were never built to detect. The problem is not volume but behavior. Each model-assisted commit carries its own blind spots. These include excessive dependencies, missing context, and incomplete validation that break the assumptions behind current AppSec programs.

Excessive dependencies

AI coding assistants often import third-party packages without verifying origin or trust level. Studies show that AI-generated code includes more than twice as many external dependencies as human-written code. This expands the attack surface and creates blind spots across dependency trees that scanners rarely catch.

Context-blind logic

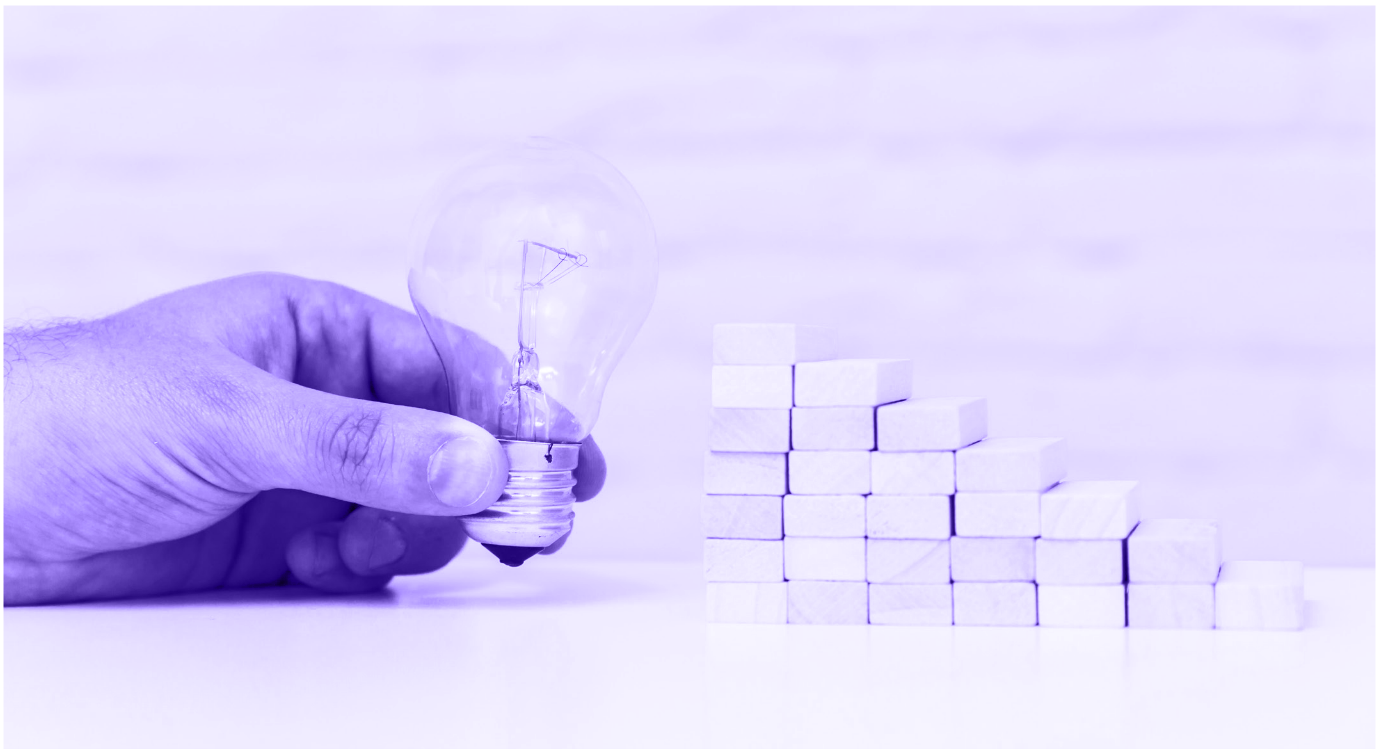
AI can produce code that looks secure but behaves incorrectly once deployed. It might reuse a public API authentication pattern inside a private service that handles sensitive data. The output passes static checks but violates policy. Human developers apply judgment; AI models do not.

Incomplete validation

AI-generated code often assumes the "happy path." Edge cases and failure conditions go untested. A 2025 review of AI-generated patches on SWE-bench found that forty-three percent fixed the primary issue but introduced new failures under adverse conditions. Automated tests passed; adversarial tests did not.

Security leaders describe similar challenges. Many report discovering more AI-generated code than expected and spending three times longer fixing related vulnerabilities because developers lack context for how the code was created.

These problems show that the challenge is structural. Security built for human intent cannot interpret AI-generated logic. The gap between code creation and code assurance keeps widening. This gap sets the stage for the emergence of agentic remediation, in which AI begins to participate in its own defense.



The Emergence of Agentic Remediation

Agentic remediation represents the next stage of application security. It moves the focus from alerting to correction. Traditional tools raise tickets and wait for responses. Agentic systems act. They detect vulnerabilities, generate candidate fixes, validate them, and explain the reasoning behind every change. This process restores confidence in codebases that now include logic no one can fully trace to a human author. However, autonomy does not negate the need for human oversight. Critical questions remain for security teams, such as: 'Which policy guardrails will your team set before agents patch production code?' This reinforces the importance of shared accountability and governance in managing AI-driven security solutions.

In research terms, an agent is a system that observes its environment, decides on an action, and executes it toward a defined goal. In security, that same model applies. These agents detect, evaluate, and correct vulnerabilities through continuous feedback and self-validation.

Autonomous detection and repair
Agentic platforms watch repositories and pipelines for AI-generated code. When they find a vulnerability, they generate a fix and validate it through layers of testing, including static analysis, integration checks, and fuzzing. Each action is logged with a clear explanation, forming an auditable record.

Recursive validation

Early automation tools worked in single passes. Agentic remediation adds feedback. One agent proposes a fix, another tests it, and a third confirms that no new risk was introduced. The system repeats this process until the fix holds. This loop closes the failure gap that earlier AI-driven patching helped create, weakening security over time.

Code-to-cloud context

True remediation depends on context. These platforms connect code-level findings to build pipelines, runtime telemetry, and cloud environments. Security teams can then focus on vulnerabilities that are actually exploitable in production rather than those that look risky in isolation.

Agentic remediation does not replace human oversight. It extends it. Security engineers set the rules, and the system works within those limits. The result is faster remediation, better accuracy, and a clear trail of reasoning that can withstand audit.

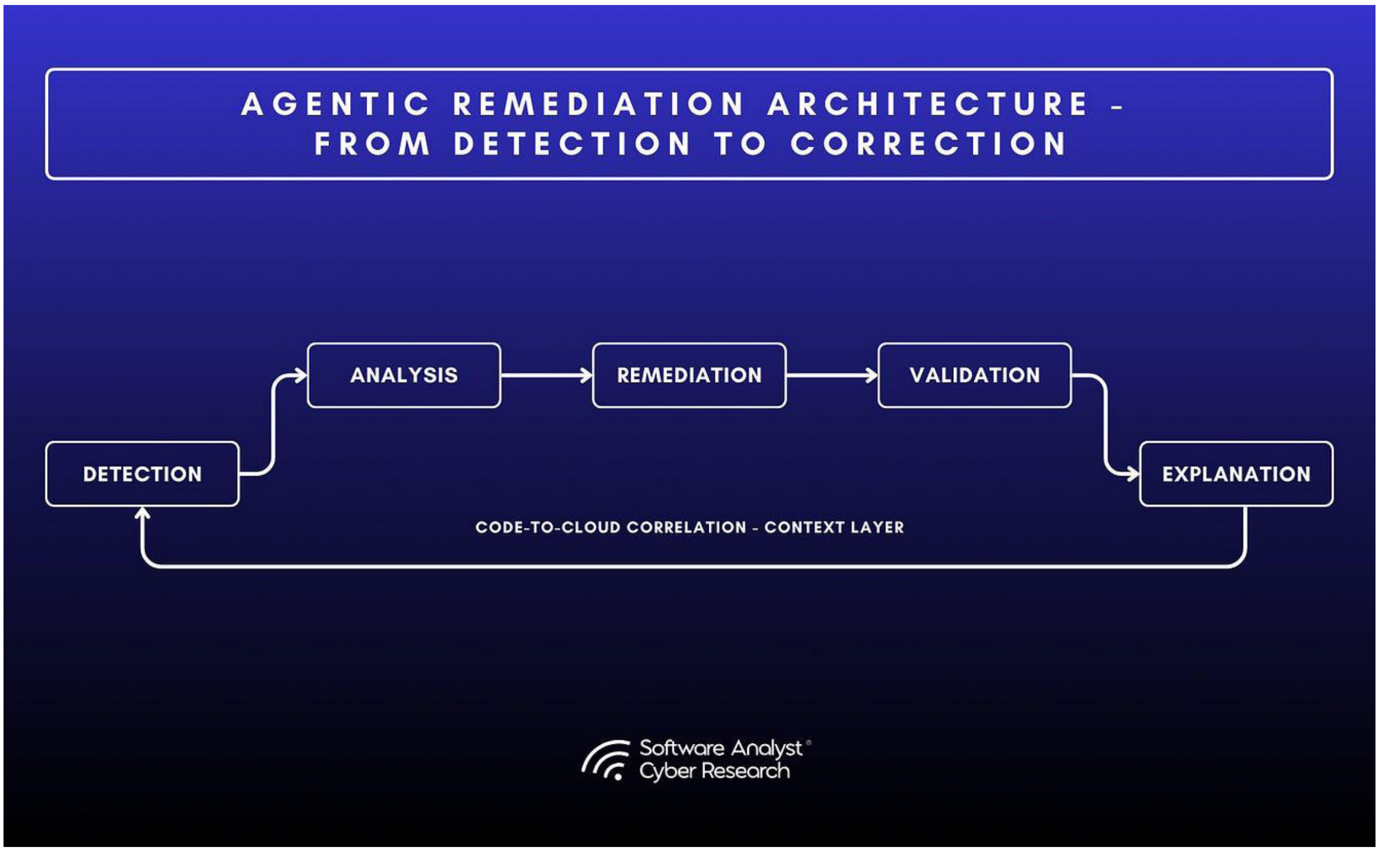
Provenance and explainability

Accountability in AI-generated code depends on traceability. Some platforms now use provenance bills of materials (PBOMs) to track AI-generated code from commit to deployment. Each change is signed, hashed, and linked to its origin model, allowing compliance teams to audit both code lineage and model influence. Other emerging approaches extend this concept through AI bills of materials (AI-BOMs) that map generated components and enforce policies to prevent unauthorized or opaque model use.

Operational outcomes

Enterprises piloting agentic remediation report measurable gains in remediation speed and accuracy. In controlled environments, validation frameworks improve successful patch rates from 67 percent to over 90 percent while cutting false positives by more than half. Developer trust grows as fixes arrive with clear reasoning rather than opaque diffs. Many organizations save about 20 engineering hours per week by reducing manual code reviews. The return on investment is tangible and immediate.

Agentic remediation does not replace human oversight. It augments it. Security engineers define the guardrails, and the system performs within them. This approach turns AI from a source of risk into an active participant in code assurance. The following section examines how leading vendors are implementing these capabilities and what differentiates their approaches.


AGENTIC REMEDIATION ARCHITECTURE - FROM DETECTION TO CORRECTION
DETECTION
ANALYSIS
REMEDIATION
VALIDATION
EXPLANATION
CODE-TO-CLOUD CORRELATION - CONTEXT LAYER
Software Analyst Cyber Research

Core Components: Multi-Agent Architecture

Agentic remediation platforms rely on multiple agents that work together to detect, fix, and verify vulnerabilities. Each agent performs a specific role, discovering AI-generated code, analyzing context, generating and validating fixes, and documenting the reasoning behind every change. Working in sequence, they form a feedback loop that maintains accuracy and accountability across the code-to-cloud lifecycle.

This design reflects a broader move toward autonomous security operations. Rather than one large engine running in isolation, specialized agents check and balance one another. Discovery provides visibility. Validation confirms quality. Explanation rebuilds trust by showing why each action was taken.

The value comes from collaboration, not complexity. Distributing responsibilities prevents single points of failure and allows each cycle through the loop to improve the next. Together, the agents create a live remediation ecosystem that becomes more accurate over time.

Vendor Analysis: How Leading Platforms Operationalize Agentic Remediation

The following analysis shows how agentic remediation is working in production. Two vendors, ***OX Security*** and ***Legit Security***, were selected for their maturity, technical depth, and alignment with the principles outlined earlier in this report. Both have released enterprise platforms that use AI to detect, validate, and correct vulnerabilities across the code-to-cloud lifecycle. Together, they represent the clearest view of how agentic remediation is being applied today.

LEGIT
SECURITY

Legit Security

Legit Security briefed SACR on its upcoming release of *VibeGuard*, a capability designed to secure AI-assisted software development and expand its AI-native ASPM platform. The company positioned 2025 as the point where AppSec must evolve from human controls to machine guardrails. Its goal is to make AI-generated code secure at creation rather than after deployment.

Legit identified three main problems driving this work: AI-generated code that lacks security training, AI assistants that create IT and supply-chain risks through excessive permissions, and the opportunity to use AI to accelerate remediation. The company showed how VibeGuard, which is available as a module within our ASPM platform or can be procured and deployed as a standalone solution (that is, without additional ASPM modules), addresses these issues by embedding a security layer inside AI coding assistants. Through an IDE plugin, VibeGuard adds secure coding guidelines, enables AI-driven scans of generated code, and offers one-click fixes. The result is a shift from finding vulnerabilities to preventing them during generation. The briefing covered the company's full ASPM stack, from traditional pipeline visibility to IDE-level AI governance through VibeGuard.

Architecture and Platform Direction

Legit's ASPM platform automatically discovers assets, pipelines, and configurations across the SDLC. It correlates findings from multiple scanners, builds dependency lineage, and identifies root causes behind recurring vulnerabilities. The result is a unified view of application risk.

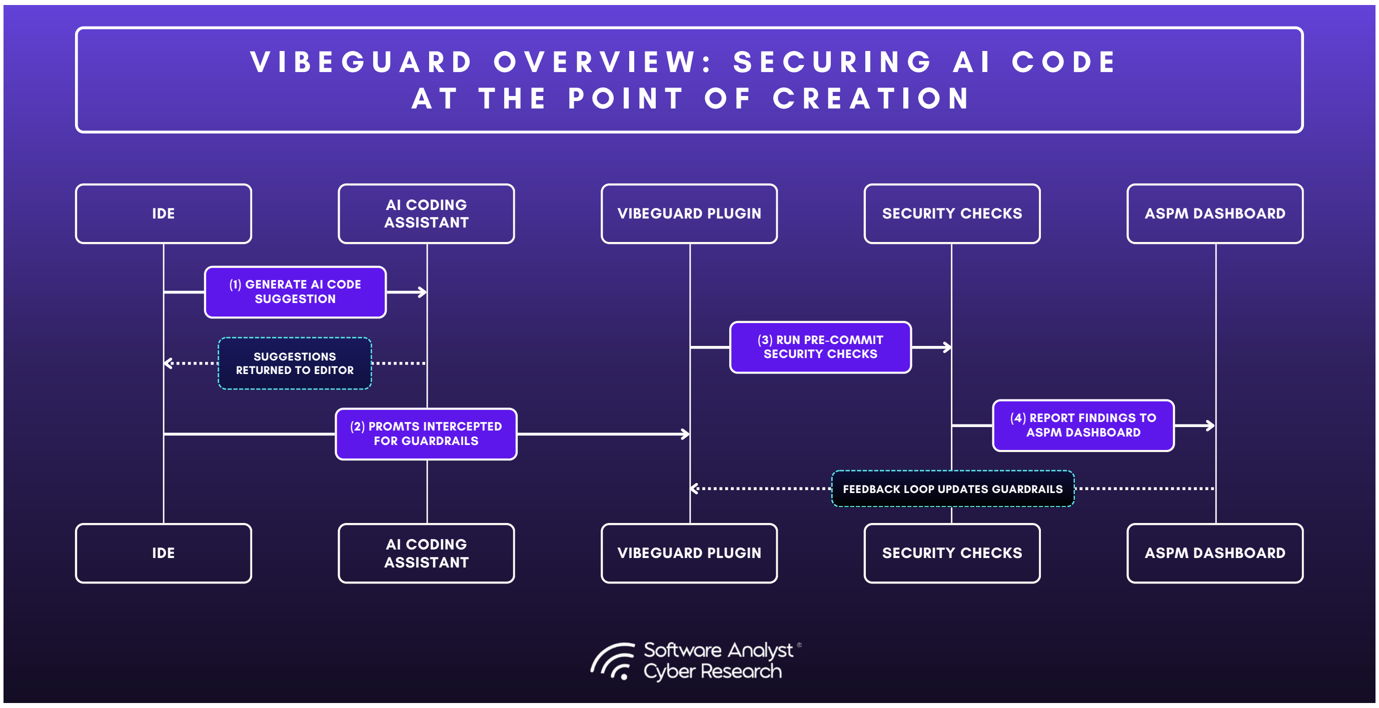
VibeGuard extends this platform into the IDE, where AI coding agents interact with developers. The system includes three layers of control:

Secure-by-instruction coding that trains AI models on organization-specific security guidelines before code generation.

Automated scanning that allows the agent to validate its own output against SAST, SCA, and secret-detection checks.

Post-generation validation that highlights newly introduced vulnerabilities and enables the AI to fix them on demand.

Each action is logged for audit purposes. CISOs gain visibility into which AI models are used, what prompts were issued, and how code was validated. This structure supports governance goals under frameworks such as SLSA and NIST SSDF.


VIBEGUARD OVERVIEW: SECURING AI CODE AT THE POINT OF CREATION
IDE
AI CODING ASSISTANT
VIBEGUARD PLUGIN
SECURITY CHECKS
ASPM DASHBOARD
(1) GENERATE AI CODE SUGGESTION
SUGGESTIONS RETURNED TO EDITOR
(2) PROMTS INTERCEPTED FOR GUARDRAILS
(3) RUN PRE-COMMIT SECURITY CHECKS
(4) REPORT FINDINGS TO ASPM DASHBOARD
FEEDBACK LOOP UPDATES GUARDRAILS
Software Analyst Cyber Research

Runtime and Research

Legit's research team focuses on threats linked to AI coding agents. They have found vulnerabilities such as *CamoLeak* and *GitLab prompt injection*, which show how compromised agents can expose data or inject malicious code. The company tracks over twenty thousand AI model components and scores them for risk and compliance.

This intelligence feeds the platform's risk scoring and model governance. Legit also helps customers build internal libraries of secure prompts and coding standards which can be distributed through VibeGuard across developer environments.

By combining runtime telemetry, IDE insights, and model intelligence, Legit maintains a continuous feedback loop between code creation, validation, and policy enforcement.

Legit helps enterprises build and distribute internal libraries of secure prompts and coding standards through VibeGuard, extending governance across developer environments.

Developer Workflow and Automation

VibeGuard integrates directly into developer workflows without slowing them down. It operates as an IDE plugin that flags insecure code in real time, suggests fixes, and checks for policy compliance before commits. Developers can remediate through chat-based commands while AppSec teams track exceptions and approvals.

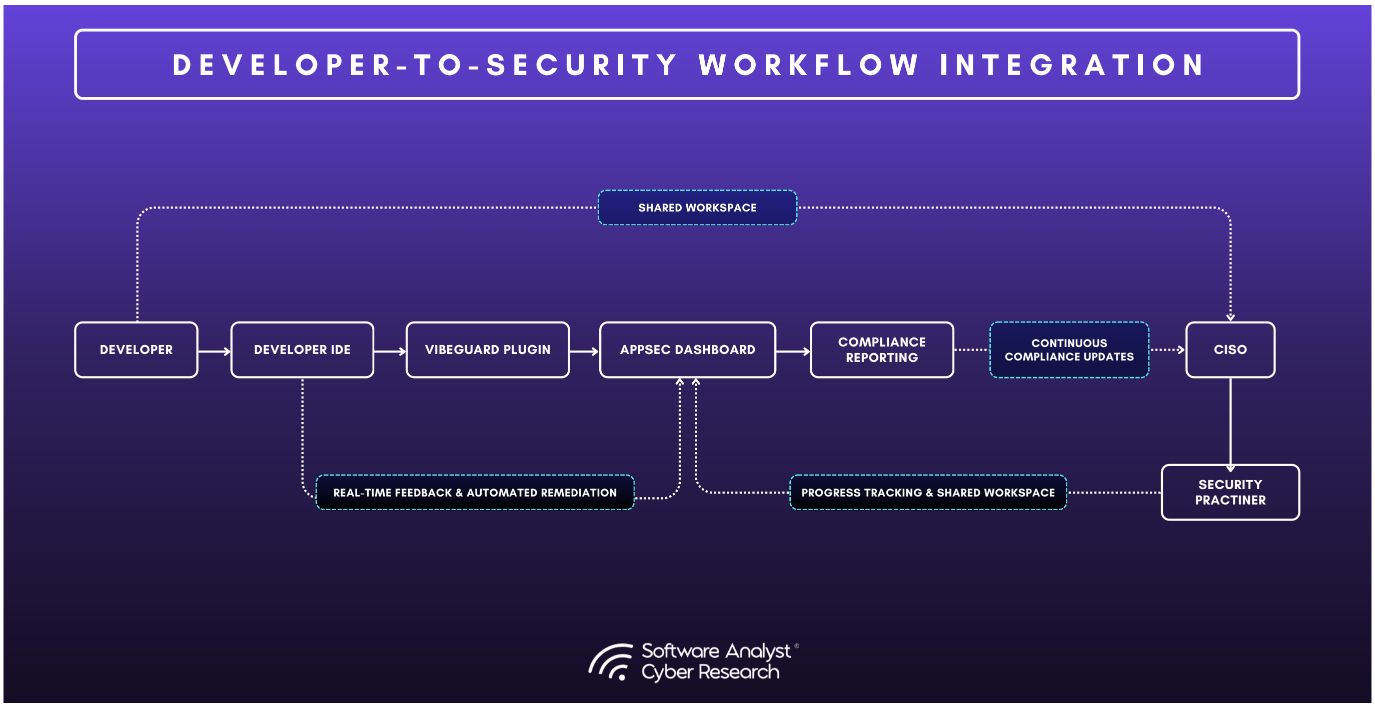
Legit also introduced AppSec Remediation Campaigns, sprint-style programs that assign ownership, deadlines, and metrics for fixing priority issues. These campaigns replace scattered tickets with measurable progress, giving teams clear visibility into MTTR and compliance results.

Together, these capabilities align developer usability with CISO accountability, showing how guardrails can coexist with speed and flexibility.

VibeGuard integrates directly into developer workflows without disrupting velocity. It operates as an IDE plugin that highlights insecure code in real time, offers fix suggestions, and prompts for policy compliance before commits. Developers can remediate through chat-based commands, while AppSec teams maintain oversight of activity and exceptions.

Legit also introduced AppSec Remediation Campaigns, structured sprint-style efforts that assign ownership, SLAs, and metrics for fixing priority issues. These campaigns replace scattered tickets with measurable progress, helping teams report on MTTR and compliance outcomes.

Together, these capabilities bridge developer usability with CISO accountability, showing how guardrails can coexist with speed and flexibility.


DEVELOPER-TO-SECURITY WORKFLOW INTEGRATION
SHARED WORKSPACE
DEVELOPER
DEVELOPER IDE
VIBEGUARD PLUGIN
APPSEC DASHBOARD
COMPLIANCE REPORTING
CONTINUOUS COMPLIANCE UPDATES
CISO
REAL-TIME FEEDBACK & AUTOMATED REMEDIATION
PROGRESS TRACKING & SHARED WORKSPACE
SECURITY PRACTINER
Software Analyst Cyber Research

Analyst Perspective

Legit Security approaches AI AppSec through governance and control rather than scanning volume. The company addresses the operational gap created by AI-assisted coding inside enterprise development environments. VibeGuard meets that need inside the IDE, enforcing guardrails, educating the agent, and preventing insecure code from leaving the developer's workspace.

This approach aligns with what CISOs are demanding: evidence of control, visibility into AI use, and traceable policies for how AI-generated code is produced. Legit's connection between governance and code creation delivers that visibility in a way that current ASPM tools do not. Its strength lies in practical discovery, code-level enforcement, and faster remediation that avoids development friction.

The next test is scale and measurement. Proving consistent MTTR reduction and maintaining performance in large, diverse environments will define how VibeGuard fits into enterprise pipelines. Among the vendors briefed for this report, Legit stands out for combining governance, visibility, and developer experience into a continuous, explainable model of guardrails for AI-driven software delivery.

Strategic Outlook

Application security is being rewritten around validation, evidence, and explainability. Scanning and ticketing alone are no longer enough. What emerges instead is a continuous system of control where AI helps teams see, prove, and fix risk in real time.

Legit Security proves that AI can be governed from the moment code is written. Together, they signal a market moving from reactive defense to active assurance, from code review to code accountability.

For CISOs, the path forward is clear. The future of AppSec is not about finding every flaw. It is about proving that the right ones were fixed, that the reasoning is traceable, and that AI now acts as the connective, predictive layer unifying code, pipeline, and runtime security into a single, explainable system.

Other Vendors Addressing AI-Generated Code Security

More AppSec vendors are adding AI features to handle AI-generated code. Snyk now uses DeepCode AI to detect vulnerabilities and suggest fixes inside developer workflows. Checkmarx has built AI-assisted remediation into the Checkmarx One platform. Veracode and Fortify are integrating language models into their static analysis and policy tuning engines.

These updates show progress, but they still center on detection. Most legacy vendors are extending existing scanning tools with AI features instead of rebuilding around continuous validation or autonomous remediation. In contrast, newer entrants such as Ox Security and Legit Security are redesigning AppSec for an AI-first development cycle. Their architectures treat AI as a control layer that connects generation, validation, and proof-of-fix.

CISOs should pay close attention to how deeply AI is embedded in each platform's remediation loop. The difference between AI-assisted detection and AI-led validation determines whether a system can explain its actions, demonstrate risk reduction, and operate continuously throughout the software lifecycle.

Practical Recommendations: What CISOs Should Do Now

Agentic remediation is no longer experimental; it is operational. CISOs should treat 2026 as the year to move from exploration to execution. The following five steps provide a pragmatic roadmap for governing, piloting, and scaling AI-generated code security.

Step 1: Discover AI-Generated Code in Your Environment

Visibility is the foundation of control. Most organizations still lack a reliable inventory of AI-generated code.

Actions:

- **Audit AI assistant adoption:** Identify which coding tools are in use (Copilot, Cursor, CodeWhisperer, Tabnine) and who has access to them. Quantify usage rates by team or repository.
- **Implement AI-BOM tracking:** Deploy discovery tools such as Legit Security or OX Security's PBOM scanning to locate AI-generated code across repositories. Tag new commits with AI metadata going forward.
- **Define governance:** Establish policies that specify approved AI tools, review requirements, and documentation standards for AI-generated code.

Step 2: Assess Current Remediation Workflows

Before automation, understand your baseline. Evaluate how effectively your current AppSec stack handles AI-generated code.

Actions:

- **Track MTTR differences**: Measure mean time to remediation (MTTR) for AI-generated versus human-written code. A two- to three-times gap signals structural inefficiency.
- **Gauge developer confidence:** Survey engineers and ask, "How confident are you in fixing vulnerabilities in code you did not write?"
- **Quantify false positives:** High alert noise from AI-authored code often masks real risk and wastes triage cycles.

Step 3: Pilot Agentic Remediation on Non-Critical Systems

Prove value before scaling. Start small and safe.

Actions

- **Select a low-risk application:** Focus on a system with known vulnerabilities and AI-generated code.
- **Evaluate vendors:**
- Remediation success rates (percentage of vulnerabilities fixed without regression)
- Developer experience and clarity of AI-generated pull requests
- Integration complexity within your CI/CD pipeline
- **Adopt Level 2 remediation:** Begin with semi-autonomous pull requests for human review before advancing to full autonomy.

Step 4: Establish Validation Frameworks

Agentic remediation succeeds only when supported by rigorous validation. This prevents iterative degradation and maintains trust.

Actions

- **Implement multi-layer validation:** Require AI-generated fixes to pass:
- Static analysis (SAST, SCA)
- Unit and integration testing
- Security-specific fuzzing
- Manual review for critical paths
- **Define rollback protocols:** Ensure rollback and monitoring procedures are in place if AI-generated patches fail in production.
- **Audit AI rationales:** Review the operational explanations behind each autonomous fix. Evaluate clarity, consistency, and recurring failure patterns.

Step 5: Scale Gradually Across the Organization

Once pilot results validate performance, expand systematically.

Actions

- **Broaden scope:** Move from pilot to departmental rollout, then to mission-critical systems.
- **Increase autonomy:** Progress from semi-autonomous (Level 2) to fully autonomous (Level 3) remediation for low-risk vulnerabilities.
- **Integrate with CI/CD:** Embed agentic remediation into existing pipelines so fixes occur before deployment.
- **Train the team:** Educate developers and AppSec analysts on how to audit, interpret, and override AI-driven fixes.

Evaluation Framework:
How to Choose a Platform

Selecting a platform for AI-generated code security requires clear evaluation criteria. Many vendors now claim "agentic remediation," but few deliver true autonomy, explainability, and depth of validation. CISOs should assess platforms across five core capabilities, vendor-specific criteria, and clear red flags.

Core Capabilities to Assess

AI Code Discovery

- Can the platform identify AI-generated code across existing repositories?
- Does it support AI-BOM tracking to meet compliance and audit requirements?
- Can it distinguish between different AI tools (for example, Copilot versus CodeWhisperer)?

Remediation Autonomy

- Does the platform support guided, semi-autonomous, and fully autonomous remediation modes?
- Can autonomy levels be configured per application or vulnerability type?
- Does it include recursive validation and self-correction to prevent degraded fixes?

Code-to-Cloud Correlation

- Can the platform link code-level vulnerabilities to runtime environments?
- Does it answer critical questions such as: Is this deployed? Is it externally exposed? Does it handle sensitive data?
- Does it prioritize vulnerabilities based on exploitability and business impact rather than solely on severity scores?

Explainability

- Does the platform generate operational rationales for each autonomous fix?
- Are the explanations clear, auditable, and defensible for compliance review?
- Can human operators override AI-driven decisions when necessary?

Validation Rigor

- Does the platform validate fixes through multiple layers, such as static analysis, dynamic testing, and fuzzing?
- What happens when validation fails? Does the system retry or escalate to human review?
- Can validation requirements be customized per application or risk class?

Vendor-Specific Questions

Selecting the right platform requires direct conversations with vendors about how their systems actually work in practice. The questions below help CISOs verify claims of autonomy, validation depth, and explainability before committing to a deployment.

1. How does PBOM track AI-generated code through the CI/CD pipeline?
2. What cryptographic standards protect PBOM integrity, such as SHA-256 or RSA-2048?
3. Can Vendor share data showing a measurable reduction in MTTR for AI-generated code remediation?
4. Are there published customer case studies with quantifiable results?

Red Flags to Watch For

- **Lack of validation frameworks.** If the platform generates fixes without rigorous multi-layer validation, you risk iterative degradation and regression.
- **Lack of explainability.** If the system cannot articulate why it made a particular change, auditability and developer trust will collapse.
- **Overstated autonomy.** Be cautious of claims promising "fully autonomous remediation." Even the most advanced systems require human oversight for high-risk or business-critical changes.
- **Missing code-to-cloud correlation.** Without runtime context, teams waste effort fixing theoretical issues while real vulnerabilities remain unaddressed.

These questions and warnings define what separates genuine agentic remediation platforms from guided automation. The following section provides SACR's analyst perspective on how to interpret these criteria in real-world evaluations.

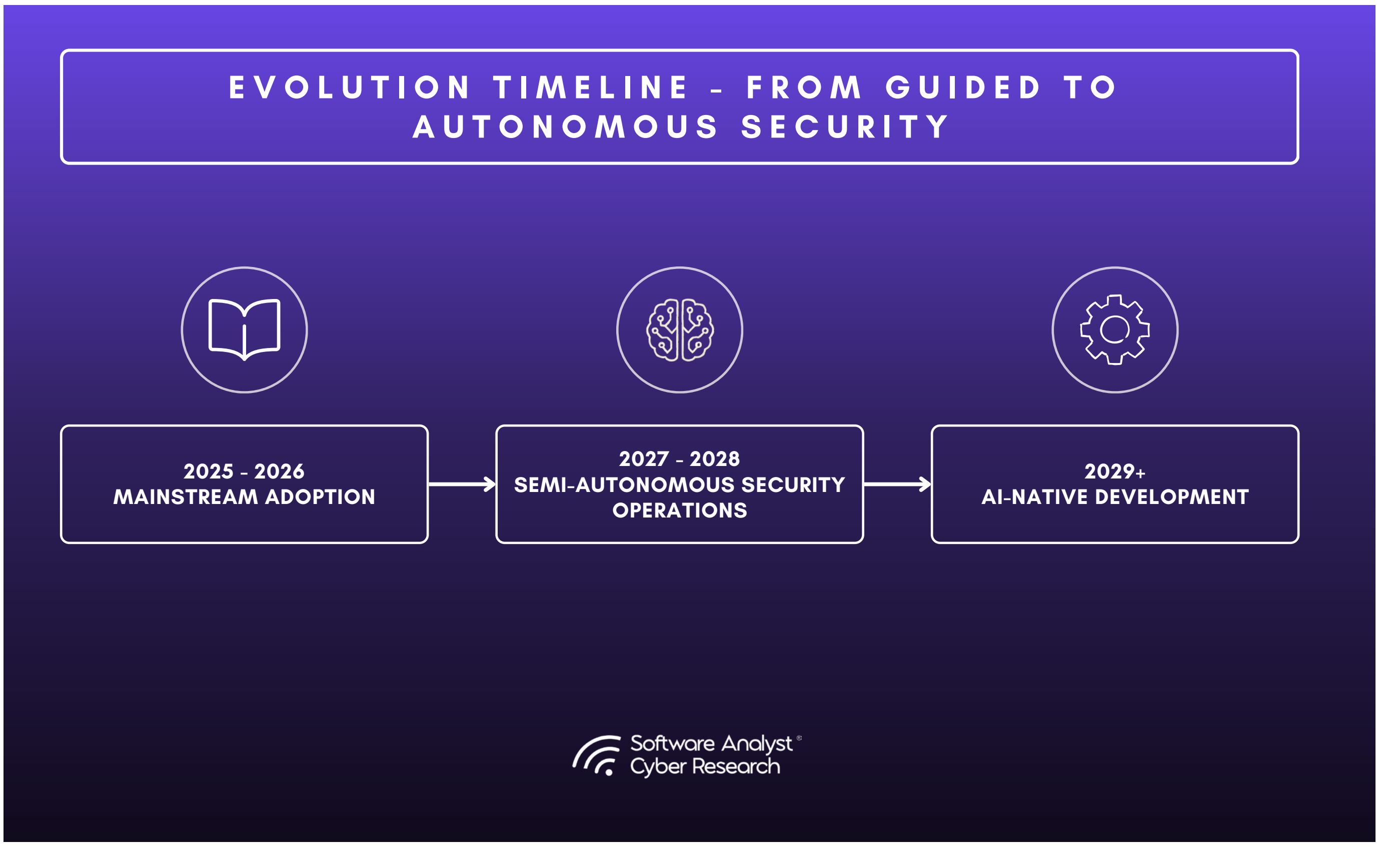
Analyst Note

CISOs evaluating this market should treat claims of autonomy with caution. Many platforms are adding AI features, but few demonstrate the discipline, validation depth, and accountability needed to operate without constant human intervention. The strongest vendors build explainability into every layer of the workflow and prove it through measurable outcomes.

What separates the credible from the experimental is behavior. A true agentic remediation platform learns from failed validations, explains its reasoning in human terms, and improves metrics that matter: MTTR, developer trust, and compliance readiness. Systems that cannot do all three remain guided by automation, not autonomy.

The lesson is simple. Agentic remediation is not about replacing people; it is about proving that AI can act with the same transparency and intent as a skilled engineer. When that happens, security becomes explainable at every layer from code to cloud.

Looking Ahead: The Future of AI Code Security


EVOLUTION TIMELINE – FROM GUIDED TO AUTONOMOUS SECURITY
2025 - 2026
MAINSTREAM ADOPTION
2027 - 2028
SEMI-AUTONOMOUS SECURITY OPERATIONS
2029+
AI-NATIVE DEVELOPMENT
Software Analyst Cyber Research

Short-Term: Mainstream Adoption

Agentic remediation will move from early adopters to mainstream enterprises. Standards for AI-BOM tracking will form under NIST and OWASP influence. Regulatory pressure from the EU Cyber Resilience Act and similar US initiatives will push organizations to document the provenance of AI code. As ROI becomes visible, developer acceptance will increase and vendor consolidation will follow.

Mid-Term: Semi-Autonomous Security Operations

Agentic systems will extend beyond vulnerabilities into compliance and predictive defense. AI agents will automatically generate audit trails and identify vulnerabilities before they are exploited. Security operations will shift from detection to prevention.

Long-Term: AI-Native Development

Human and AI coding will merge. Security agents will operate continuously within IDEs and pipelines, verifying code as it is written. Low-risk issues will be fixed autonomously, while humans oversee strategic and high-risk changes.

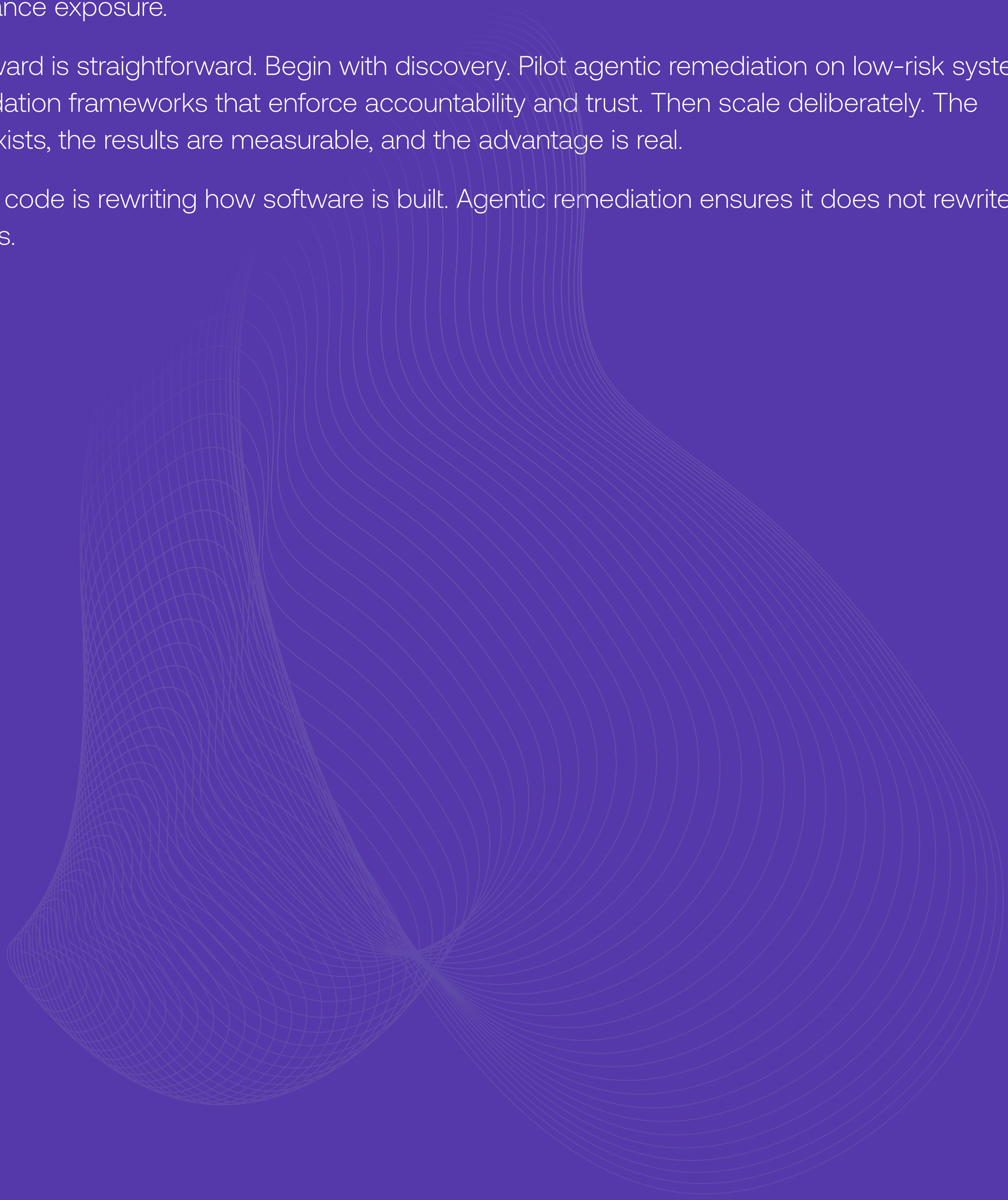
Conclusion

Agentic remediation marks a defining shift in application security. Detection-first workflows can no longer keep pace with AI-generated code. Software now writes itself faster than traditional AppSec teams can review it, and the tools built for human authorship were never designed to secure code without context.

The risk is clear. Developers cannot reliably fix what they did not create, and repeated "AI-assisted improvements" often amplify rather than eliminate vulnerabilities. Organizations that act now will maintain velocity without sacrificing control. Those who delay will inherit opaque codebases, slower remediation, and rising compliance exposure.

The path forward is straightforward. Begin with discovery. Pilot agentic remediation on low-risk systems. Establish validation frameworks that enforce accountability and trust. Then scale deliberately. The technology exists, the results are measurable, and the advantage is real.

AI-generated code is rewriting how software is built. Agentic remediation ensures it does not rewrite how security works.



Disclaimer

This report was developed by Software Analyst Cyber Research (SACR) to examine how emerging vendors are addressing the security risks of AI-generated code through agentic remediation with some sample vendors. It is intended to inform CISOs and security leaders about market direction, not to rank or endorse specific products. All vendor information reflects data provided during SACR briefings and publicly available sources as of November 2025. Readers should use these findings as guidance for evaluation and due diligence rather than as a substitute for independent assessment.

Software Analyst®
Cyber Research

business

personal

softwareanalyst.io

SACR®

Trusted research. Sharp insights. Real conversation.

CISO	VENDOR
SECURITY TEAMS	INVESTORS