LEGIT

EXECUTIVE BRIEF

Questions AI is Creating That Security Can't Answer Today

AI-assisted development is outpacing traditional AppSec governance - learn what leading security teams are doing to shore up these gaps

Executive Summary

AI-led development has moved from experimentation to standard practice across modern software teams. GitHub reports that 92% of developers now use AI coding tools, with AI-generated code contributing to 40% or more of new code in many organizations. While productivity gains are substantial – studies show 25-35% faster task completion – AI introduces governance and audit questions that traditional AppSec programs were never designed to answer.

The challenge isn't technical capability. It's architectural mismatch. Traditional security controls were built around code that humans write in controlled environments and commit to repositories. But AI generates code at the developer endpoint, before any traditional control point can see it. This creates fundamental gaps in auditability, traceability, and defensibility.

This brief examines where the traditional audit model breaks down, what questions auditors are now asking, and how leading security teams are shifting controls earlier in the SDLC to remain audit-ready and compliant.

92%	of developers now use AI coding tools
40%	of new code is AI-generated in many organizations
25–35%	faster task completion with AI coding tools

The Wake-Up Call Comes When Auditors Arrive

Consider this increasingly common scenario:

A financial services institution undergoes their annual SOC 2 Type II audit. Midway through, the auditors asked to see documentation of controls around AI-generated code. The CISO realized they couldn't answer basic questions:

- Which developers were using GitHub Copilot, ChatGPT, Claude or other AI tools?
- What percentage of production code was AI-generated versus human-written?
- Were security policies enforced before AI-generated code was committed?

Their traditional AppSec scanning and code review processes were all post-commit controls. By the time those controls activated, AI-generated code containing hardcoded credentials, license violations and other security vulnerabilities had already entered the codebase. It fact, these issues may have been lingering for days or weeks or longer.

This isn't an isolated case. Security leaders across industries report similar experience. The common thread: traditional AppSec controls were designed for a different world – a world where human wrote, understood and had full context into every line of code they produced.

The Paradigm Shift: How AI Breaks Traditional Audit Assumptions

The traditional SDLC audit model assumes human developers write code in controlled environments. In this scenario, IDEs connect to managed repositories, with security controls at commit, PR and build stages. AI breaks every one of those assumptions.

Audit Dimension	Traditional SDLC	AI-Driven SDLC
Code Authorship	Human developer	Human + AI + autonomous agents
Creation Point	Repository commit	Developer endpoint (IDE, browser)
Control Timing	Post-commit (repository, CI, runtime)	Pre-commit (generation time)
Evidence Source	Scans, code reviews, logs	Prompt context + enforcement decisions
Traceability	Who changed code (commit author)	How and why code was generated

The Critical Difference

In traditional workflows, auditors could trace code back to individual developers through commit history and code review logs. With AI, that traceability breaks down. The developer who committed AI-generated code may not have written a single line. Because of this, they lack material understanding of the context AI used to create the code. Was the context relevant and appropriate for the application or pulled from another, potentially unreliable or malicious source?

Unless controls exist at the point of generation, before code reaches the repository, there's no reliable way to reconstruct what happened or prove that policies were enforced.

The Audit Questions Security Teams Can't Answer Today

As AI adoption accelerates, auditors – whether internal or external – are asking questions that expose gaps in traditional AppSec governance:

1 Visibility & Discovery

Where is AI being used in software development?
Most organizations have incomplete visibility into AI adoption. Developers use GitHub Copilot, ChatGPT, Claude, Cursor, Windsurf, and dozens of other tools, often without formal approval. Shadow AI usage is pervasive, with one recent survey finding that 68% of developers use unapproved AI tools for work tasks.

Representative audit questions:

- What AI coding tools are authorized for use?
- How do you detect unauthorized AI tool usage?
- Which teams and projects are using AI-generated code?
- Can you quantify AI-generated code as a percentage of total codebase?

Traditional answer: "We rely on developer self-reporting and periodic surveys." This doesn't satisfy auditors.

2 Policy & Governance

What policies govern AI-generated code?
Having policies documented is table stakes, but auditors now ask whether those policies are actually enforced and how you can prove it. Many organizations have AI usage guidelines in wikis or handbooks, but no mechanism to ensure compliance.

Representative audit questions:

- What security and compliance policies apply to AI-generated code?
- How are these policies communicated to developers?
- What prevents developers from using AI to generate code that violates policy?
- Can you show evidence of policy enforcement?

Traditional answer: "We have policies in our developer handbook and security training." This doesn't demonstrate enforcement or provide audit evidence.

3 Control Enforcement

Can you prove controls were enforced before code reached production?
This is where most organizations face the most significant challenge. Traditional security controls activate in PR reviews, CI pipelines, or pre-deployment scans. That means AI-generated code containing secrets, vulnerabilities or license violations may sit in repositories for days or weeks before detection.

Representative audit questions:

- At what point are security controls enforced on AI-generated code?
- Can you prevent insecure AI-generated code from being committed?
- How do you handle secrets or credentials in AI-generated code?
- What's your mean time to detect and remediate AI-introduced vulnerabilities?

Traditional answer: "We scan repositories and run SAST in CI." But this is reactive; controls activate after risky code already entered the codebase.

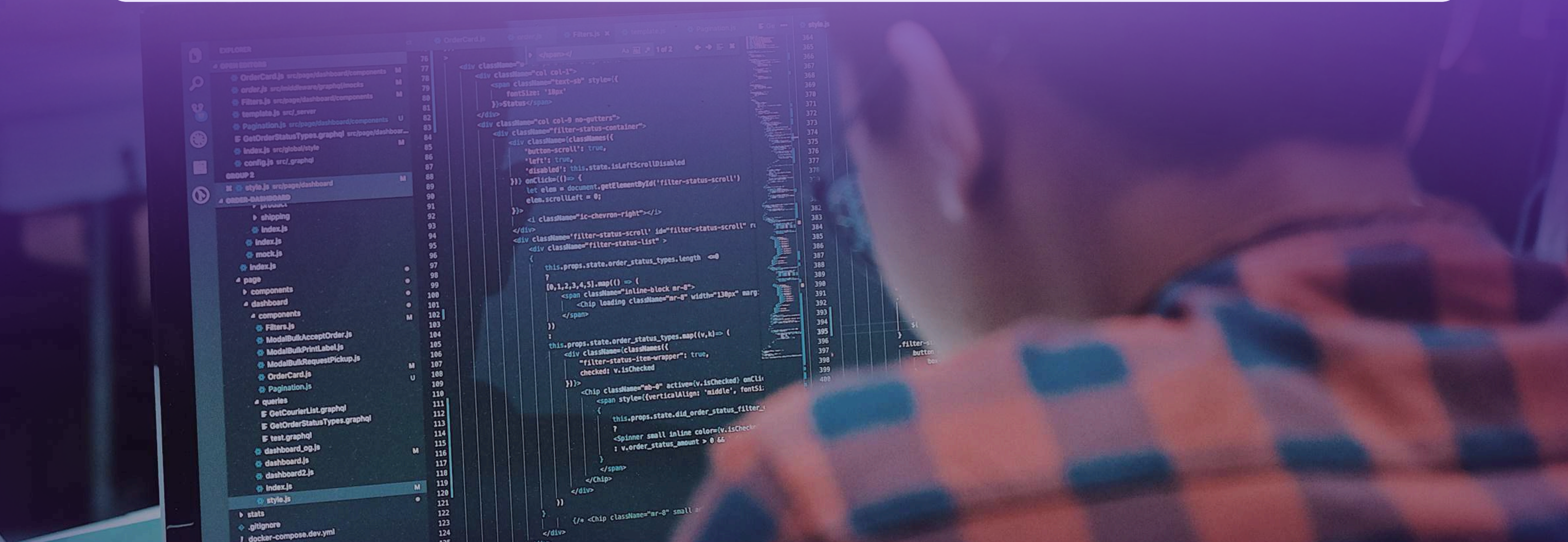
4 Traceability & Evidence

Can you reconstruct AI-assisted development decisions?
When an auditor asks, 'How was this code generated? What was the prompt? What context was provided to the AI? What policies were checked?', most organizations can't answer. Git commit logs show who committed code, but not how AI was involved or what decisions were made.

Representative audit questions:

- Can you identify which code was AI-generated versus human-written?
- Can you reproduce the context that led to the specific AI-generated code?
- What audit trail exists for AI-assisted development activities?
- If an issue is found, can you determine its origin and impact radius?

Traditional answer: "We can see commit history and code review comments." But this doesn't capture AI involvement, prompts or generation-time decisions.



The Pre-Commit Governance Model: Shifting Controls Earlier

To answer these audit questions, security controls must shift earlier in the SDLC. Security must be enforced at the moment AI generates code, before it reaches the repo. This is the pre-commit governance model.

How it works:

- **Endpoint visibility:** Security observes AI interactions at the developer endpoint (in the IDE, browser or wherever code is generated)
- **Inline policy enforcement:** Policies are evaluated in real-time as code is generated, with immediate feedback to developers
- **Automatic evidence collection:** Every AI generation event creates an audit record including prompt, context, generated code, policy evaluation, developer action
- **Developer-friendly workflow:** No extra steps required–evidence collection happens transparently as developers work

This model enables security teams to answer all four categories of audit questions:

- **Visibility:** "We have real-time visibility into all AI coding activity across our organization"
- **Governance:** "Our policies are enforced automatically at the point of code generation"
- Enforcement: "Controls prevent policy violations before code is committed to repositories"
- **Traceability:** "We maintain complete audit trails of AI-assisted development activities"

What Leading Security Teams are Doing

Organizations at the forefront of AI governance share common patterns:

1 They accept AI as inevitable, not optional

Leading teams don't try to ban or restrict AI tools. They recognize that AI-assisted development is here to stay and focus on governing it effectively rather than fighting adoption.

2 They shift security left to the endpoint

Instead of relying solely on post-commit scanning, they deploy controls at the developer endpoint where AI code generation happens. This enables prevention rather than just detection.

3 They treat audit readiness as a continuous requirement

Rather than scrambling during audit season, they maintain continuous evidence collection and can answer auditor questions in real-time with concrete data.

4 They measure outcomes, not just coverage

Leading teams track metrics like percentage of AI-generated code auto-approved under policy, mean time to policy compliance and developer satisfaction with AI governance tools.

How to Move Forward

Like it or not, believe it or not, AI adoption in software development is already happening across your organization. The question isn't whether your developers use AI to write code; it's whether you can prove to auditors, regulators and executives that you govern it responsibly.

Traditional AppSec controls were built for a world where humans wrote and understood code. AI changes the game: code is now generated at the developer endpoint, before traditional control points can see it. This architectural shift demands a governance response that matches the technology.

Teams that shift governance to the moment of AI code generation can answer the audit questions that will define modern software assurance. They can demonstrate visibility, prove policy enforcement and maintain traceability throughout the development lifecycle.

Those that don't will face increasingly difficult conversations as compliance frameworks, auditors, and security expectations catch up to AI reality. The audit questions outlined in this brief are just the beginning. As AI adoption deepens and regulatory attention intensifies, the gap between traditional controls and AI-driven development will only widen.

The time to prepare isn't during your next audit. It's now.

Next Steps

See how Legit VibeGuard enables organizations to answer audit questions with confidence.

→ Request a demo to see pre-commit AI governance in action

→ Download our AI Security Maturity Model to assess your current state

→ Read our Technical Architecture Guide for platform and security engineering teams

Build fast with AI.
Secure with
AI-powered ASPM.

Legit Security is the AppSec platform purpose-built to secure AI-powered development. Our AI-native ASPM secures modern software development, including AI-first pipelines, code assistants, agents, and vibe coding. With unmatched visibility across the SDLC and from code to cloud, Legit makes it easy to identify, prioritize, and fix AppSec issues that matter most to the business.