BEYOND THE SDLC

The Future of Security in the Age of Agentic Development

A Strategic Analysis by

AGENTIC SECURITY

2025 Edition

Executive Summary

The software development lifecycle (SDLC) as we know it was designed for a world where humans wrote every line of code. That world ended in 2023.

Today, AI coding assistants generate up to 25% of code at leading tech companies.[1] By 2027, 70% of professional developers will use AI-powered coding tools daily.[2] This shift is happening faster than most organizations can adapt their security practices.

[1] Sundar Pichai (Google CEO), cited in MIT Technology Review, 'AI Code Generation at Scale,' 2024.

[2] NetCorp, 'AI-Generated Code Statistics 2025,' 2025.

The result? A security gap that grows wider every day. Studies show 45-62% of AI-generated code contains security vulnerabilities[3]—even when using the latest models. Traditional SDLC security controls, designed for human authorship and discrete development phases, are fundamentally inadequate for this new reality.

[3] Veracode, '2025 GenAI Code Security Report,' July 2025. Study of 100+ LLMs across 80 real-world coding tasks.

"The SDLC isn't dead, but it must evolve. Security can no longer be checkpoints. It must be continuous guardrails embedded in every AI-human interaction."

This report examines the fundamental shift in software development, why traditional security approaches are failing, and what organizations must do to secure AI-augmented development without sacrificing velocity.

Key Findings:

- AI-generated code has vulnerability rates 2-3x higher than human-written code across major programming languages
- The generative AI cybersecurity market will reach $35.5B by 2031, driven by these new risks[4]
- 'Shadow AI' usage is creating massive security blind spots in organizations that haven't established governance[5]
- Traditional code review fails when reviewers don't understand the context that generated the code
- New attack vectors—prompt injection, hallucinated dependencies, over-privileged agents—require entirely new threat models

[4] Research and Markets, 'Generative AI Cybersecurity Research Report 2025-2030,' November 2025.

[5] SC Media, 'Cybersecurity in 2025: Agentic AI to change enterprise security,' January 2025.

Section 1: The Shift

How Development Changed Overnight

In November 2022, the release of ChatGPT set off a chain reaction that would fundamentally alter software development. Within months, every major tech company rushed to integrate AI coding assistants into their workflows. GitHub Copilot, Cursor, Replit, and dozens of other tools promised to dramatically accelerate development velocity.

They delivered on that promise. Developers report 30-50% productivity gains when using AI coding tools. Google's CEO confirmed that 25% of the company's code is now AI-assisted.[6] The velocity gains are real and irresistible.

[6] Sundar Pichai (Google CEO), cited in MIT Technology Review, 'AI Code Generation at Scale,' 2024.

But velocity without security is just speed toward failure.

What Changed?

- **Authorship became blurred:** When AI generates code, who's responsible? The developer who prompted it? The AI vendor? The organization?
- **Context windows replaced structured phases:** Development is now a continuous conversation, not discrete stages of design, code, test, deploy.
- **Code review assumptions broke:** Traditional review assumes the author understands what they wrote. AI-generated code violates this assumption.
- **Velocity trumped scrutiny:** The productivity gains are so compelling that security concerns get deprioritized.
- **New attack surfaces emerged:** Prompt injection, hallucinated dependencies, and over-privileged agents created threat categories that didn't exist before.

Section 2: The Threat Landscape

The Vulnerability Explosion

In July 2025, Veracode released the most comprehensive study on AI-generated code security to date.[7] The findings were sobering: across 100+ large language models and 80 real-world coding tasks, 45% of AI-generated code contained security vulnerabilities aligned with the OWASP Top 10.

[7] Veracode, '2025 GenAI Code Security Report,' July 2025. Study of 100+ LLMs across 80 real-world coding tasks.

45-62%

of AI-generated code contains vulnerabilities[8]

[8] Veracode, '2025 GenAI Code Security Report,' July 2025. Study of 100+ LLMs across 80 real-world coding tasks.

72%

security failure rate for Java[9]

[9] Ibid.

Even more concerning: these vulnerability rates show no improvement trend. As models get better at generating syntactically correct code, they're not necessarily getting better at generating secure code.

New Attack Vectors

Beyond traditional vulnerabilities, AI-augmented development introduces entirely new categories of risk:

- **Prompt Injection:** Malicious actors can manipulate AI outputs through carefully crafted prompts, causing the AI to generate backdoors or vulnerabilities.
- **Hallucinated Dependencies:** AI models sometimes reference packages or libraries that don't exist. Attackers exploit this through 'slopsquatting'—creating malicious packages with these hallucinated names.
- **Over-Privileged Agents:** Agentic coding tools that can autonomously commit code, access APIs, or modify infrastructure create new privilege escalation risks.
- **Context Window Poisoning:** Long development sessions accumulate context that can influence future code generation in unpredictable ways.
- **Credential Exposure:** AI tools trained on public code repositories may suggest patterns that inadvertently expose credentials or sensitive data.

These aren't theoretical risks. Security researchers have demonstrated all of these attacks in controlled environments. It's only a matter of time before they're weaponized at scale.

Section 3: Why Traditional Security Fails

The SDLC's Fundamental Assumptions

The modern Software Development Lifecycle emerged in the 1960s and has been refined over decades. Its security controls are built on several fundamental assumptions:

- Developers understand the code they write
- Code authorship is clear and attributable
- Development happens in discrete, sequential phases
- Code review can catch issues because reviewers understand context
- Security gates at phase transitions are sufficient
- Threat models are relatively stable over the project lifecycle

Agentic development violates every single one of these assumptions.

Where Traditional Controls Break Down

Code Review Becomes Superficial

Traditional code review assumes the reviewer can assess whether code does what it should and doesn't do what it shouldn't. But when AI generates 100 lines of code in response to a 10-word prompt, reviewers often lack the context to evaluate it properly. They see syntactically correct code that appears to solve the problem—but miss subtle security implications.

Security Gates Are Too Infrequent

SDLC security gates—design review, pre-commit checks, staging deployment reviews—happen at discrete intervals. But agentic development is continuous. Code is generated, tested, regenerated, and refined in rapid iteration. By the time code reaches a security gate, it may have been through dozens of AI-assisted modifications.

Threat Models Are Static

Organizations conduct threat modeling exercises at the beginning of projects, identifying risks based on architecture and requirements. But agentic development introduces dynamic risk—the threat surface changes with every AI interaction. A threat model that doesn't account for prompt injection, hallucinated dependencies, or agent privilege escalation is fundamentally incomplete.

Accountability Gets Lost

When a vulnerability ships to production, traditional processes ask: 'Who wrote this code?' With AI-assisted development, the answer is murky. Was it the developer who prompted the AI? The AI model vendor? The organization that configured the tooling? This ambiguity isn't just philosophical—it has real implications for liability, insurance, and regulatory compliance.

Section 4: The Future State

From Gates to Guardrails

The solution isn't to abandon the SDLC or to ban AI coding tools. The solution is evolution: adapting proven security principles for a new development paradigm.

"Security must shift from periodic checkpoints to continuous guardrails embedded in every human-AI interaction."

This means security controls that operate in real-time, configured directly into AI tooling, validated continuously rather than periodically, and designed specifically for AI-generated code patterns.

The Four Pillars of Agentic Security

Based on extensive research and real-world implementation, we've identified four essential pillars for securing AI-augmented development:

1. Threat Model Assessment

Comprehensive threat modeling that accounts for AI-specific attack vectors including prompt injection, hallucinated dependencies, over-privileged agents, and context window manipulation. Organizations must understand what they're defending against before they can defend effectively.

2. Security Control Integration

Embedding security controls throughout the development process: pre-commit hooks that scan AI-generated code, SAST/DAST tools configured for AI patterns, dependency scanning for hallucinated packages, audit trails for AI-generated sections, and policy enforcement with separation of duties.

3. Developer Education & Enablement

Training developers on secure AI-assisted development practices: prompt engineering for security, understanding AI limitations and hallucinations, session hygiene and context management, code review techniques for AI output, and tool-specific best practices.

4. Configuration Optimization

Optimizing AI tool configurations for security: .cursorrules, .claude, .aider files configured with security-first prompts, custom system prompts that emphasize security requirements, organization-wide policy enforcement, and governance frameworks for configuration management.

Together, these four pillars create a comprehensive framework—the Agentic Code Security Framework (ACSF)—that organizations can implement to secure AI-augmented development without sacrificing velocity.

What Mature Agentic Security Looks Like

By 2026-2027, organizations that successfully adapt will have:

- **AI-specific threat models** that are living documents, updated as new tools and attack vectors emerge
- **Security prompts embedded directly into AI tooling,** ensuring every code generation starts with security requirements
- **Real-time scanning and validation** that flags vulnerabilities as they're generated, not days later in staging
- **Clear governance frameworks** defining approved tools, configurations, and usage policies
- **Developer training programs** that treat secure AI coding as a core competency
- **Audit trails that track AI-generated code** separately, enabling faster incident response and compliance
- **Metrics and monitoring** for AI code quality, vulnerability rates, and tool effectiveness

These organizations will maintain the velocity advantages of AI-augmented development while dramatically reducing security risk. They'll be able to confidently answer security questionnaires, pass audits, and win enterprise customers who demand robust security practices.

Section 5: Getting Started

Five Questions Every CISO Should Ask Today

1. Do we have visibility into AI coding tool usage across our organization?

If you don't know which tools developers are using, you can't assess risk. Conduct a survey or audit to identify all AI coding assistants in use, including both approved and 'shadow AI' tools.

2. Does our threat model account for AI-specific attack vectors?

Review your most recent threat model. Does it mention prompt injection, hallucinated dependencies, or over-privileged agents? If not, it's incomplete for the current threat landscape.

3. Can we identify which code in our repositories was AI-generated?

Without audit trails distinguishing AI-generated from human-written code, you can't analyze patterns, respond to incidents, or demonstrate compliance. Implement tagging or commit message conventions now.

4. Do our developers know how to review AI-generated code securely?

AI-generated code requires different review techniques than human-written code. If you haven't trained your team specifically on this, they're likely missing vulnerabilities.

5. Are our AI tools configured with security requirements?

Most developers use default configurations that prioritize functionality over security. Have you established organization-wide policies and optimal configurations for each tool in use?

First Steps: A Roadmap

Organizations beginning their agentic security journey should follow this prioritized roadmap:

Immediate (Next 30 Days):

- Audit current AI coding tool usage across all development teams
- Establish basic governance: which tools are approved, what data they can access
- Begin tagging AI-generated code in commits for future traceability

Short-term (Next 90 Days):

- Update threat model to include AI-specific attack vectors
- Implement pre-commit hooks that flag common AI-generated vulnerabilities
- Develop initial security configuration standards for approved tools
- Conduct pilot training for one development team on secure AI coding

Medium-term (6-12 Months):

- Deploy comprehensive ACSF framework across all development teams
- Integrate AI-aware SAST/DAST tools into CI/CD pipeline
- Establish metrics and monitoring for AI code quality and security

- Achieve initial compliance readiness (SOC 2, ISO 27001, etc.)

The organizations that move quickly and decisively will turn agentic security from a liability into a competitive advantage—demonstrating to customers and partners that they can harness AI's productivity gains without compromising security.

Conclusion: The Choice Ahead

The shift to AI-augmented development is not optional. Organizations that resist will find themselves unable to compete on velocity with those that embrace it. But embracing AI coding tools without adapting security practices is equally untenable—it trades long-term security debt for short-term productivity gains.

The choice isn't whether to use AI coding tools. The choice is whether to secure them properly.

Organizations that implement the Agentic Code Security Framework—comprehensive threat modeling, integrated security controls, developer education, and optimized configurations—will achieve something remarkable: they'll maintain the velocity advantages of AI while dramatically reducing risk.

"The future of software development isn't human or AI—it's human and AI, working together with security built into every interaction."

The SDLC isn't dead. It's evolving. The question is whether your security practices will evolve with it.

Ready to Secure Your AI-Augmented Development?

Agentic Security helps organizations implement the ACSF framework through comprehensive assessments, hands-on implementation, and ongoing support. Our team has secured 100% AI-generated applications and helped enterprises achieve compliance readiness while maintaining development velocity.

Schedule a consultation to discuss:

- Comprehensive assessment of your current AI coding tool usage and security posture
- Custom threat modeling for your specific environment and risk profile
- Implementation roadmap for the ACSF framework
- Developer training programs tailored to your tech stack and tools
- Ongoing support and optimization as the threat landscape evolves

Contact Agentic Security

Email: contact@agenticsecurity.so

Web: agenticsecurity.so

References

1. Sundar Pichai (Google CEO), cited in MIT Technology Review, 'AI Code Generation at Scale,' 2024.

2. NetCorp, 'AI-Generated Code Statistics 2025,' 2025.

3. Veracode, '2025 GenAI Code Security Report,' July 2025. Study of 100+ LLMs across 80 real-world coding tasks.

4. Research and Markets, 'Generative AI Cybersecurity Research Report 2025-2030,' November 2025.

5. SC Media, 'Cybersecurity in 2025: Agentic AI to change enterprise security,' January 2025.

6. Sundar Pichai (Google CEO), cited in MIT Technology Review, 'AI Code Generation at Scale,' 2024.

7. Veracode, '2025 GenAI Code Security Report,' July 2025. Study of 100+ LLMs across 80 real-world coding tasks.

8. Veracode, '2025 GenAI Code Security Report,' July 2025. Study of 100+ LLMs across 80 real-world coding tasks.

9. Ibid.