LEGIT

The AI Security Maturity Model for AI-First Development Teams

A framework for evolving from reactive cleanup to proactive AI governance & protection

Introduction

AI adoption in software development often, if not always, moves faster than security programs can adapt to keep pace with. This creates a predictable and problematic pattern: teams start using AI informally, security discovers usage reactively and organizations scramble to establish governance after risks have materialized.

This maturity model provides a roadmap for evolving AI security from reactive incident response to proactive, audit-ready governance. It's designed to help teams adopting AI-led development – whether in the early stages or further along in use across engineering – understand where they are today, what good looks like and next steps to progress your program.

How to use this model

- Read each stage description and identify where your organization is today
- Use the self-assessment questions to confirm your current stage
- Focus on progressing one stage at a time – don't try to skip stages
- Reassess quarterly as AI workflows and tools evolve

The Five Stages of AI Security Maturity

The maturity model consists of five stages, each representing a distinct level of AI security capability:

Stage	Description	Primary Risk	Control Point	Focus Area
1. Unmanaged	Informal AI use, no visibility or policies	Unknown exposure	N/A	Visibility
2. Reactive	Issues found late, high rework cost	Reactive cleanup burden	Repository	Detection
3. Visible	Monitoring exists, inconsistent enforcement	Policy drift, gaps	PR checks	Coverage
4. Proactive	Inline guardrails prevent issues	Residual edge cases	Pre-commit	Prevention
5. Operationalized	AI-native security, continuous improvement	Minimal, well-managed	Multi-layer	Optimization

Stage 1

Unmanaged

Stage Status: Shadow AI usage, no visibility, reactive incident response

Characteristics

- Developers use AI tools (e.g., GitHub Copilot, ChatGPT, Claude, Cursor) without centralized visibility or approval; tools may be procured individually vs. enterprise licenses
- No formal policies governing AI-generated code
- Security team learns about AI usage reactively (often through incidents)
- No tracking or visibility into code by AI vs. humans
- No policy and guardrail enforcement mechanisms at generation time
- Leadership likely unaware of AI adoption extent

Common Challenges

"We don't know what we don't know"	Unknown exposure to AI-introduced risks
Shadow AI adoption	Engineering teams use tools without telling security
Incident-driven awareness	Security first learns about extend of AI usage when hardcoded credentials, license violation, or vulnerabilities are discovered in production
Compliance risk	Cannot answer auditor questions about AI governance

Stage 1

Unmanaged

What Success Looks Like

At the end of Stage 1, you should have:

- Complete inventory of AI tools in use across development teams
- Basic policy documentation (even if not yet enforced)
- Leadership awareness and buy-in that AI governance is needed
- Plan to move to Stage 2 within 90 days

Self-Assessment Questions

- ☐ Do you know which developers are using AI coding assistants?
- ☐ Do you have documented policies for AI-generated code?
- ☐ Can you tell auditors what percentage of your code is AI-generated?
- ☐ Have you had any security incidents related to AI-generated code?
- ☐ Can you prove to auditors that AI-generated code complies with security policies?

If 2+ answers are "No": You're likely in Stage 1

Next Steps to Stage 2

- **Survey development teams** on AI tool usage (what tools, how often, for what tasks)
- **Document current state and risks** identified in the survey
- **Draft basic AI code generation policies** covering secrets, licenses and critical vulnerabilities
- **Establish executive sponsorship** for AI governance program
- **Timeline:** 1-3 months to move to Stage 2

Stage 2

Reactive

Stage Status: Post-commit detection, high rework cost, reactive remediation

Characteristics

- AI usage policies are documented and communicated
- Traditional security tools (SAST, SCA) scan repositories and CI
- Issues are found after commit, which requires context switching and rework
- Security team reacts to findings through tickets, Slack alerts, email and other manual processes
- Mean time to remediation: 3-7 days for most issues

Common Challenges

Developer friction	Late feedback breaks flow and creates resentment toward security
High rework cost	Issues found hours/days later require significant effort to fix
Alert fatigue	Volume of findings overwhelms developers and security
Incomplete visibility	Can only see AI-generated code after it's committed

What Success Looks Like

- Automated scanning of all repositories for AI-introduced issues
- Baseline metrics established (issues found, time to remediate, etc.)
- Clear SLAs for remediation based on severity
- Security incidents reduced significant compared to Stage 1

Stage 2

Reactive

Self-Assessment Questions

- ☐ Do you have automated scanning for AI-generated code issues?
- ☐ Can developers see security findings within 24 hours of commit?
- ☐ Do you track metrics on AI-related security issues?
- ☐ Are remediation SLAs met the strong majority of the time?

If 3+ answers are "Yes": You're in Stage 2

Next Steps to Stage 3

Implement repository-level monitoring to track AI tool usage patterns

Deploy pre-commit hooks for secrets detection and basic policy checks

Establish coverage goals (e.g., % of repositories with AI scanning enabled)

Timeline: 2-4 months to move to Stage 3

Stage 3

Visible

Stage Status: Comprehensive monitoring, inconsistent enforcement, policy drift

Characteristics

- Real-time visibility into AI tool usage across the organization
- Monitoring deployed across most repositories and teams
- Some pre-commit controls (secrets detection, basic policy checks)
- Enforcement is inconsistent as different teams use different tools/approaches
- Policy drift: written policies don't always match what's enforced

Common Challenges

Tool sprawl	Multiple scanning tools with overlapping coverage and conflicting results
Coverage gaps	Some teams/repos not monitored; new AI tools adopted without vetting
Policy drift	What's documented differs from what's enforced
Audit challenges	Can show monitoring but struggle to prove consistent enforcement

What Success Looks Like

- Nearly complete coverage of repositories, teams and AI tools
- Centralized dashboard showing AI usage, policy violations trends
- Pre-commit hooks preventing most issues before they reach repository
- Mean time to remediation reduced from days to minutes/hours

Stage 3

Visible

Self-Assessment Questions

- ☐ Do you have visibility into 90%+ of AI tool usage?
- ☐ Are pre-commit controls deployed across most teams?
- ☐ Can you demonstrate policy enforcement to auditors?
- ☐ Do developers receive security feedback before code is committed?

If 3+ answers are "Yes": You're in Stage 3

Next Steps to Stage 4

Deploy endpoint-level controls (IDE integrations with security controls, at-generation identification of vulnerabilities and issues) for real-time enforcement

Standardize tooling across teams to eliminate coverage gaps and policy drift

Shift focus from detection to prevention by catching issues at generation time

Timeline: 3-6 months to move to Stage 4

Stage 4

Proactive

Stage Status: Inline guardrails, pre-commit prevention, developer-friendly enforcement

Characteristics

- Security controls enforced at the point of code generation (AI IDE, endpoint)
- Developers receive immediate inline feedback; issues caught in seconds, not hours
- Most security issues prevented before commit
- Automated evidence collection for all AI-generated code
- Developer satisfaction with security tools improves significantly

Common Challenges

Edge case handling	Residual risks in complex scenarios not yet covered by policies
Policy refinement	Continuous tuning needed to reduce false positives
Scaling challenges	Maintaining performance as organization grows

What Success Looks Like

- Nearly all AI-generated code auto-approved under policy
- Mean time to policy compliance: < 5 minutes
- Developer rework hours reduced by 60% (minimum)
- Full audit trail of all AI-assisted development activities
- Security team can answer all auditor questions with data

Stage 4

Proactive

Self-Assessment Questions

- [ ] Are security controls enforced at the developer endpoint?
- [ ] Do developers receive inline feedback within seconds of AI code generation?
- [ ] Are almost all issues prevented before reaching the repository?
- [ ] Have developer satisfaction scores with security tools improved?

If 3+ answers are "Yes": You're in Stage 4

Next Steps to Stage 5

Implement continuous policy optimization using ML to reduce false positives and improve accuracy

Build feedback loops from production incidents back to generation-time policies

Establish center of excellence for AI security best practices

Timeline: 6-12 months to move to Stage 5

Stage 5

Operationalized

Stage Status: AI-native security, continuous improvement, industry-leading maturity

Characteristics

- Multi-layer defense: endpoint + repository + CI + runtime controls
- AI-powered policy optimization and threat detection
- Continuous improvement based on production feedback
- Full auditability and compliance automation
- Security is invisible to developers–operates transparently at high velocity
- Organization becomes thought leader in AI security practices

Common Challenges

Maintaining edge	Staying ahead of rapidly evolving AI capabilities and risks
Scaling excellence	Extending best practices to acquisitions and new teams
Innovation risk	Ensuring governance doesn't stifle experimentation with new AI tools

What Success Looks Like

- Close to 100% automated policy enforcement
- Zero unplanned security incidents from AI-generated code
- Audit preparation time reduced
- Developer velocity increased significantly compared to Stage 1
- Security team recognized as enabler of innovation, not blocker

Stage 5

Operationalized

Self-Assessment Questions	
	☐ Do you have multi-layer AI security controls across the SDLC?
	☐ Are policies automatically optimized based on production feedback?
	☐ Can you produce complete audit reports in < 1 hour?
	☐ Have you eliminated unplanned AI-related security incidents?

If 3+ answers are "Yes": You're in Stage 5

Continuous Improvement	
	Share best practices with industry peers and security community
	Pilot emerging AI security technologies before general availability
	Contribute to industry standards and frameworks for AI security
	Timeline: Ongoing refinement and optimization

The good news

Most teams can progress from Stage 1 to Stage 3 within 6 months with the right approach and tooling. Movement from Stage 3 to Stage 4 typically takes 3-6 months, representing the shift to proactive, inline enforcement.

Value Delivered by Stage Progression

As teams mature through the stages, measurable improvements emerge:

Stage 1 → Stage 2

- Reduce security incidents from AI code
- Establish baseline metrics for tracking progress
- Create policy foundation for future enforcement

Stage 2 → Stage 3

- Reduce mean time to remediate
- Increase policy coverage to majority of teams and repositories
- Prevent most issues before they reach repositories

Stage 3 → Stage 4

- Prevent the vast majority of issues before commit (vs. Stage 3)
- Reduce developer rework hours
- Improve developer satisfaction with security
- Achieve full audit readiness with automated evidence

Stage 4 → Stage 5

- Increase automation to nearly 100%
- Eliminate unplanned AI-related security incidents
- Reduce audit preparation time
- Achieve improvement in overall developer velocity

GETTING STARTED

Your First 90 Days

Regardless of your current stage, here's how to begin improving AI security maturity:

Week 1-2

Assess Current State

- Use this maturity model to identify your current stage
- Survey development teams on AI tool usage and pain points
- Document quick wins and critical gaps

Week 3-4

Build Foundation

- Draft initial AI security policies (start simple)
- Secure executive sponsorship and budget
- Select 1-2 pilot teams for initial implementation

Week 5-8

Pilot Implementation

- Deploy monitoring or enforcement tools with pilot teams
- Gather feedback and refine approach
- Document learnings and adjust policies

Week 9-12

Scale and Measure

- Roll out to broader organization
- Establish metrics dashboard
- Plan next quarter improvements to progress to next stage
- Reassess maturity and celebrate progress

CONCLUSION

The Path Forward

AI accelerates software development, but without mature governance, that speed creates risk. The teams that succeed don't fight AI adoption– they evolve how they govern it.

This maturity model provides a practical roadmap. Whether you're in Stage 1 struggling with shadow AI, or Stage 3 working toward proactive controls, the path forward is clear: focus on one stage at a time, measure progress, and continuously improve.

The organizations that invest in AI security maturity today will be the ones that can move fastest with confidence that governance enables velocity rather than constraining it.

The question isn't whether to mature your AI security practices. It's how quickly you can progress to the next stage.

Next Steps

Ready to advance your AI security maturity?

Request a maturity assessment to determine your current stage and create a personalized roadmap

See how Legit VibeGuard can support pre-commit AI governance and code security

Download the Executive Brief on AI audit challenges for security leadership

Read the Technical Architecture Guide for platform 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.