

AI CODE GUARDRAILS:

**A PRACTICAL GUIDE FOR
SECURE ROLLOUT**

snyk

INTRODUCTION

LEARN HOW TO ROLL OUT AI CODING TOOLS LIKE GITHUB COPILOT AND GEMINI CODE ASSIST SECURELY WITH PRACTICAL GUARDRAILS, USAGE POLICIES, AND IDE-BASED TESTING.

Tools like GitHub Copilot and Google Gemini Code Assist help teams generate code at scale, reduce boilerplate, and speed up delivery, resulting in unprecedented boosts in productivity. But with greater speed comes greater security risk. Studies show that 27% of AI-generated code contains vulnerabilities, reflecting volume and velocity, not tool failure.

To manage that risk without losing momentum, organizations need to implement security guardrails and checks and controls that prevent AI-generated code from introducing vulnerabilities into production.

This guide offers a practical framework to help engineering leaders and security teams roll out AI assistants safely and scalably, using Snyk's platform to help reinforce AI governance policies. From pull request checks to IDE scanning and conditional access policies, each section outlines real implementation tactics you can adopt today to start building your AI-readiness, without compromising developer productivity.

ENFORCE GUARDRAILS AT THE PULL REQUEST STAGE

Why it matters: Pull requests are a natural place to catch AI-generated vulnerabilities before they reach production.

Before fully rolling out AI coding assistants, it's important to ensure your development process includes automated security checks. These guardrails help prevent risky code from being merged into your main branch, and pull requests are the most logical place to start.

With [Snyk's Pull Request \(PR\) checks](#), you can scan every code change as it's submitted, flagging issues early and integrating security into the review process without disrupting workflows.

You can also use the [Snyk CLI](#) in your CI/CD process as a second checkpoint for more mature pipelines. This layered approach helps maintain consistency across teams and deployment paths.

Catching issues here is a meaningful win, but it often comes after code has been written, reviewed, and maybe even tested. Fixing those issues can create additional overhead. That's why, in the next section, we'll look at how to move these checks even earlier in the development lifecycle.

SHIFTING LEFT: AVOIDING AI-GENERATED CODE INEFFICIENCIES

Why it matters: Catching security issues during development reduces rework and keeps developers focused on building, not backtracking.

Since Snyk's earliest days, we've emphasized the importance of identifying vulnerabilities as early as possible, ideally while the code is still being written. That philosophy remains especially important as teams begin using AI code assistants.

While pull request checks catch risky code before it's merged, they come after the work is done. By then, developers may have already built functionality on top of insecure logic, so fixing a simple bug could require refactoring larger components.

Instead, we recommend extending your guardrails directly into the development environment. Using the Snyk IDE plugin, developers can get real-time feedback as they code, catching vulnerabilities before the code ever leaves their editor.

For teams working in agentic environments, like Cursor or GitHub Copilot chat-based workflows, the same level of scanning can be achieved using the Snyk local MCP server, which runs security checks in the background as code is generated.

Shifting left doesn't just improve security posture, it reduces friction for developers and accelerates delivery. And when those guardrails feel like part of the flow, adoption becomes much easier, which is what we'll explore next.

REQUEST EVIDENCE OF LOCAL SECURITY TESTING

Why it matters: Verifying security setup at the start encourages responsible tool use and builds good security habits early.

Before granting developers access to AI coding assistants, consider implementing a lightweight access requirement: proof that local security testing is in place, preferably in the IDE, where issues can be identified and fixed immediately.

One option is to ask developers to upload a screenshot showing that they have installed the Snyk security IDE plugin and attest that they will proactively test their AI-assisted code locally.

For example, developers can upload a screenshot showing that the Snyk IDE plugin is installed and confirm that they'll proactively test AI-generated code during development.

Teams working in agent-based environments (like Cursor or Copilot) can alternatively connect to the Snyk local MCP server, which supports agent-driven workflows and scans AI output as it's created.

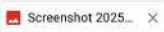
As a secondary layer, organizations can still use pull request checks to catch issues before merging. For even greater efficiency, Snyk Agent Fix enables autonomous remediation by suggesting secure alternatives in context, further streamlining the development experience.

Code Assistant Access Request Form

Complete this form to request access to an AI coding assistant. Include a screenshot demonstrating that you have installed a [Snyk IDE plugin](#) to test code locally.

Upload a screenshot showing that the Snyk IDE plugin is installed for local testing *

Upload 1 supported file: PDF or image. Max 10 MB.

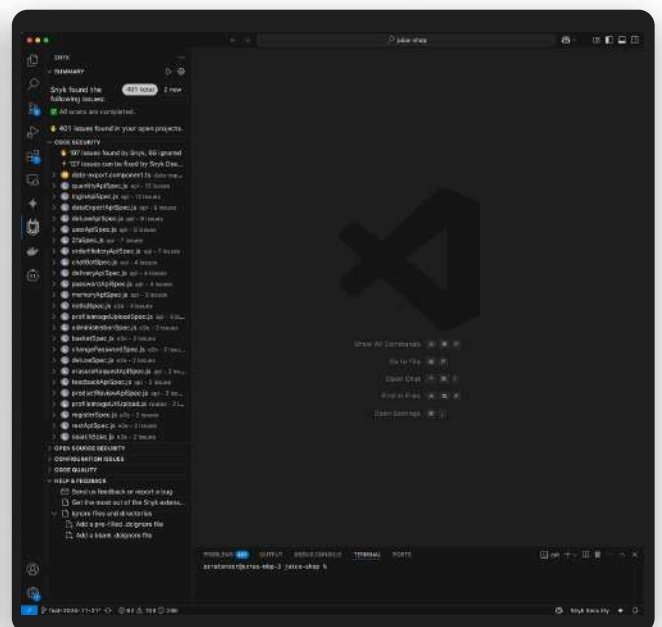

Screenshot 2025...

Provide any additional context on the request

Requesting access to accelerate development

By submitting this form, I attest that I will only use the AI coding assistant in conjunction with the Snyk IDE plugin.

Submit
Clear form



Example evidence showing the installation of the Snyk security IDE plugin



AUDIT EXISTING USAGE AND ONBOARDING NEW TEAMS

Why it matters: Visibility into tool usage helps ensure guardrails are working and that they are adopted where it counts.

If AI coding tools are already used across your organization, it's not too late to implement secure practices. Conduct periodic audits to identify any blind spots where developers may be using AI coding assistants without local security checks.

Use [Snyk's Developer IDE and CLI usage reports](#) alongside your AI coding assistant's admin console to cross-reference who's actively using assistants, and whether security tooling like the IDE plugin is also in place.

Gemini Access Report

Name	Email	License Assigned	Last Active	Last Detected Snyk Scan
John Smith	john.smith@snyk.io	2025-01-15	2025-04-15	2025-04-16 15:04:31.154
Jane Jones	jane.jones@snyk.io	2025-01-15	2025-02-22	
Danial Hill	danial.hill@snyk.io	2025-02-14	2025-04-16	

For a more scalable approach, Snyk Essentials provides centralized visibility into developer adoption of key security tools, helping platform and security teams track IDE plugin usage, identify gaps (e.g. missed scans), and monitor adoption trends over time.

A simple "trust but verify" model can go a long way. Some teams send automated reminders or light-touch enforcement notices, letting developers know that their access may be paused if security tools are missing or inactive.

INTEGRATE SECURITY AWARENESS INTO DEVELOPER TRAINING

Why it matters: Developers are best positioned to prevent vulnerabilities introduced by AI-generated code, but they can only do so if they understand the risks.

As AI tooling becomes part of everyday development, security training should evolve accordingly. Ensure that developer onboarding and continuing education explicitly cover the risks of AI-generated code, and reinforce the importance of local testing as a first line of defense.

[Snyk Learn](#) includes a targeted lesson on the [OWASP Top 10 for LLM and GenAI](#), helping teams understand emerging threats and adopt safer AI practices.

Explore our whitepaper, [Developer Training in Cybersecurity](#) for a broader perspective on secure development upskilling.

Quiz

Test your knowledge!



Quiz

What must you do if you want access to an AI code assistant tool?

- ☐ Include "be secure" in your prompts
- ☐ Install and use the Snyk IDE plugin
- ☐ Download a code assistant from the web

Keep Learning

- AI generated code is not immune to security vulnerabilities.
- It is your responsibility to test code locally and in security gates.

Example of developer education: Snyk Learn quiz

PROACTIVE TOOLING AND ACCESS CONTROL

Why it matters: When access to AI tools is tied to secure configurations, you create guardrails that scale and ensure security isn't optional.

For organizations with more centralized control over developer environments and automated distribution, there's an opportunity to deploy security tooling alongside access to AI code assistants.

There are several ways to approach access management, but how you choose will ultimately depend on your tools, how you use them, and your company culture.

For example, if your company utilizes endpoint management systems, you could consider allowing listing access to AI code assistants for users who have demonstrated installation of local security testing tools or recently confirmed their commitment to security practices. If you're using tools like Microsoft Intune, Jamf, or Citrix, you might configure dynamic domain access rules that grant access to Gemini, Copilot, Cursor, or Windsurf only after a developer has met the defined security prerequisites.

If your development teams leverage virtual development environments, access to coding assistants can be granted programmatically in conjunction with the Snyk IDE plugin. See the following example of [dev container](#) setup granting Microsoft Copilot and Snyk extensions in VS Code:

```
None
{
  "image":
  "mcr.microsoft.com/devcontainers/typescript-node",
  "forwardPorts": [3000],
  "customizations": {
    // Configure properties specific to VS Code.
    "vscode": {
      // IDs of extensions to install when the container is
      created.
      "extensions":
      ["snyk-security.snyk-vulnerability-scanner",
      "github.copilot"]
    }
  }
}
```

THE PATH FORWARD: SECURE INNOVATION

AI-assisted development is no longer experimental — it's already changing how teams write, test, and ship code. But with this speed and scale comes risk, and it's up to engineering and security leaders to ensure those risks don't derail progress.

Guardrails are the key. When implemented early in IDEs, agents, PRs, and access workflows, they allow developers to move faster, not slower. They remove barriers by embedding security into the development experience itself.

Whether your teams are just starting to explore AI tooling or are already rolling it out across environments, the practices in this guide offer a practical framework for building trust in that process without introducing unnecessary friction.

Secure innovation isn't just possible, it's operational. And Snyk is here to help build trust in your AI. [Talk to our team to get started!](#)

Want to learn more about how
Snyk builds trust in AI software?

EXPLORE SNYK NOW.

snyk