Guide

Quick Guide: 5 Steps to Tackle Shadow Code Now

Security, Compliance, and Platform Stability Start with Visibility

Your greatest vulnerability in high-velocity software delivery environments may be hiding in plain sight, or worse, completely invisible.

This is called shadow code: unapproved, unreviewed, and untracked code that finds its way into your CI/CD pipelines outside the standard guardrails of your software delivery lifecycle. This risk is far from hypothetical. Nearly 45% of AI-generated code, even from advanced models, contains security flaws such as SQL injection and cross-site scripting.

It can come from anywhere: AI-assisted development tools like Copilot or ChatGPT, one-off manual pipeline tweaks, forgotten scripts left by former team members, unregistered runners, or even third-party integrations that were never formally onboarded.

Once it’s in your system, shadow code can quietly undermine security, compliance, and platform stability, while building operational debt that may not be uncovered until it’s too late.

In this guide, we’ll break down:

- What shadow code is and why it’s dangerous to your CI/CD pipelines
- How to identify shadow code in your current systems
- Practical steps to control and eliminate existing shadow code
- Strategies to prevent shadow code from creeping back in

CloudBees

What is Shadow Code?

Shadow code is any build logic, automation, or script that runs in your software delivery process but operates outside standard review, approval, or security scanning controls. It's not inherently malicious, but it's inherently dangerous, because it's invisible to governance and often bypasses organizational safeguards.

It most often appears when:

- AI-assisted tools write config files, CI jobs, or scripts without code review
- Manual changes are made directly in the build or deployment pipelines
- Team turnover leaves behind orphaned assets like runners, credentials, or shell scripts
- Unapproved third-party tools are connected to your CI/CD without security validation

Why It Matters

You can't govern what you can't see. Left unchecked, shadow code becomes a silent multiplier of risk that can:

Expose sensitive data	by leaking credentials or tokens in unmonitored scripts
Introduce unstable dependencies	that cause silent build breaks and runtime failures
Create compliance blind spots	by bypassing audit trails and mandated reviews
Increase operational complexity	as security and engineering teams spend cycles untangling the mess
Weaken supply chain security	by introducing unverified tools, unmanaged runners, or hidden integration points that attackers can exploit

CloudBees

5 Steps to Tackle Shadow Code Now

The five steps provide a security-first framework for finding hidden build logic, enforcing governance, and maintaining continuous assurance across your delivery pipelines.

Step 1
Inventory All Build and Pipeline Assets

Why it matters:

You can’t secure what you can’t see. Shadow jobs, orphaned runners, and unmanaged credentials expand your attack surface and weaken auditability.

Take action:

- Catalog every pipeline (active and dormant) along with custom scripts, CLI tools, service accounts, runners, jobs, and dependencies. Many teams are shocked to learn how many artifacts are running without active ownership.
- Use your CI/CD platform or specialized tools like CloudBees Unify and CloudBees CI to automatically discover and map jobs, credentials, runners, and their relationships.

Pro Tip:

Run a full “shadow asset sweep” at least once per quarter to ensure your inventory stays current and defensible during audits.

Step 2
Implement AI-Aware Developm ent Policies

Why it matters:

Generative AI is now part of modern development, but it changes the threat landscape. AI tools can produce code that can bypass peer review and security scanning, creating invisible vulnerabilities.

Take action:

- Require developers to label AI-assisted contributions in pull requests
- Require teams to peer review any AI-generated code before merging it.
- Maintain an “Approved AI Tools” list that clearly defines where and how AI can be used

Pro Tip:

Create accountability without blocking innovation. Bake AI-use disclosure into your workflow for transparency without slowing delivery.

CloudBees

Step 3

Embed Guardrails Directly in Your Pipelines

Why it matters:	Security policies only work if they're enforced automatically, at scale. Manual reviews can't keep up with tool sprawl.
Take action:	• Implement policy-as-code to block unauthorized jobs, tokens, API calls, or dependencies • Enforce registry and image checks to reduce supply change risk
Pro Tip:	Use tools like OPA and CloudBees to enforce policy-as-code directly in your pipelines, catching rogue jobs before they can weaken your delivery chain.

Step 4

Continuously Monitor for Invisible Changes

Why it matters:	Pipelines often look static in Git, but in reality, they evolve dynamically - making \shadow code nearly invisible.
Take action:	• Monitor jobs appearing outside version control, detached branches with deployment permissions, and code paths skipping scans • Unify secrets monitoring and drift dashboards to surface risks early
Pro Tip:	Treat drift detection as a control point - continuous monitoring reduces mean time to detect (MTTD) for hidden vulnerabilities.

Step 5

Educate and Empower Your Developers

Why it matters:	Security isn't effective in isolation. Preventing shadow code requires a culture where developers own security outcomes alongside delivery speed.
Take action:	• Host a "Shadow Code Awareness" workshop and add PR checklists • Give developers a clear reporting path for rogue jobs or AI-generated contributions • Empower engineers to report rogue jobs or AI-classified code
Pro Tip:	Position security as an enabler, not a blocker. Celebrate developers who flag shadow code, making secure practices part of how they innovate.

CloudBees

The Bottom Line: Seal Hidden Vulnerabilities in Your Software Supply Chain

Shadow code isn’t an edge-case problem. It is the next serious blind spot in the software supply chain. The cost of ignoring it will always outweigh the effort of addressing it.

The good news? With the right combination of visibility tools, automated guardrails, continuous monitoring, and developer empowerment, platform teams can address this challenge without slowing delivery.

Modern solutions are evolving to help organizations gain unified visibility across their software delivery ecosystem, automatically detect security gaps, and streamline remediation, turning what used to be manual detective work into proactive risk management.

See these principles in action.

Watch how CloudBees Unify provides a single experience for software delivery, helping enterprises streamline development, reduce manual operational tasks, and coordinate work across different systems.

Continuously redefine what’s possible through software. Book a demo today at cloudbees.com/contact-us

---

CloudBees

CloudBees, Inc.
cloudbees.com
info@cloudbees.com