

Code Scanning in the AI Era



Weaponized AI or Machine Speed Threats



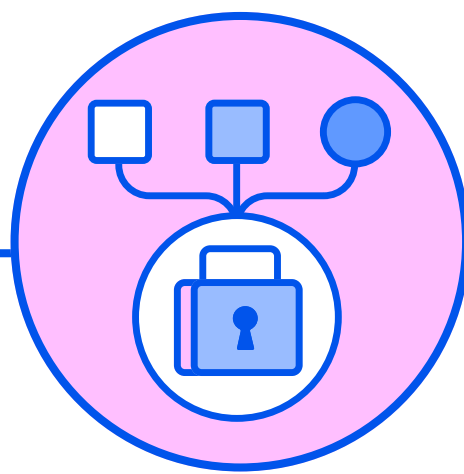
AI Writes More Code

Code generation is acceleration software delivery and expanding attack surface.



AI Finds more Bugs

Frontier models can reason about application logic and uncover vulnerabilities humans and scanners miss.



Attackers Move Faster

The gap between vulnerability discovery and exploitation is shrinking.

Why it Matters

- 1 Traditional scanning relies on predefined patterns and struggles to reason about application intent, allowing business logic flaws and auth issues to slip through.
- 2 Code security findings often lack the runtime context needed to determine exploitability, leaving developers overwhelmed by alerts that may not matter in production.
- 3 Even when vulnerabilities are identified, assigning ownership, prioritizing fixes, and driving remediation remains heavily manual and difficult to scale.

The Layered Defense: Balancing Speed, Depth & Cost

Scanning for the AI era means balancing deterministic and non-deterministic scanners

Layer 1 · Rules-Based SAST

Ongoing · Fastest · Lowest cost

Catches well-known, syntax-based flaws (SQLi, XSS). Cheap enough to run on every commit across every repo — your continuous baseline hygiene.

Layer 2 · AI-SAST Semantic Reasoning

Continuous · Moderate cost · Deeper

AI reasons about code like a security researcher to surface intent-dependent flaws — IDOR, business logic errors, broken authorization. Run on every major update; target high-value repos (internet-facing, sensitive data).

Layer 3 · Frontier Model Analysis

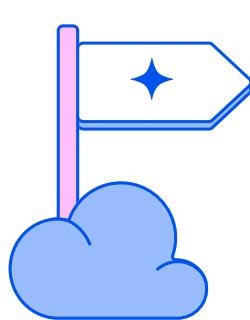
Periodic · Highest cost · Deepest

Intensive architectural scans that uncover multi-step, complex zero-days. Reserve for mission-critical apps where a breach is catastrophic; run on a risk-based cadence.

← Faster, more affordable, and syntax-level coverage

→ Slower, more costly, deeper reasoning

How to Operationalize AI Code Analysis



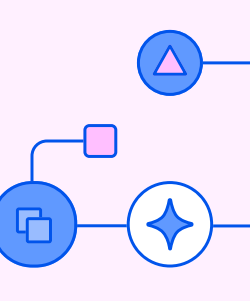
Prioritize

Prioritize the repositories with customer facing apps, internet-exposed APIs, and sensitive authentication services.



Scan

Layer fast pattern-matching SAST with AI semantic reasoning to catch the widest set of vulnerabilities, from known syntax flaws to business logic and authorization gaps.



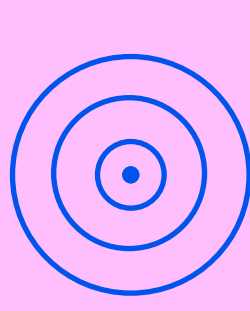
Contextualize

Enrich and correlate findings across cloud, runtime, identity, and adversarial context to confirm what's truly exploitable.



Remediate

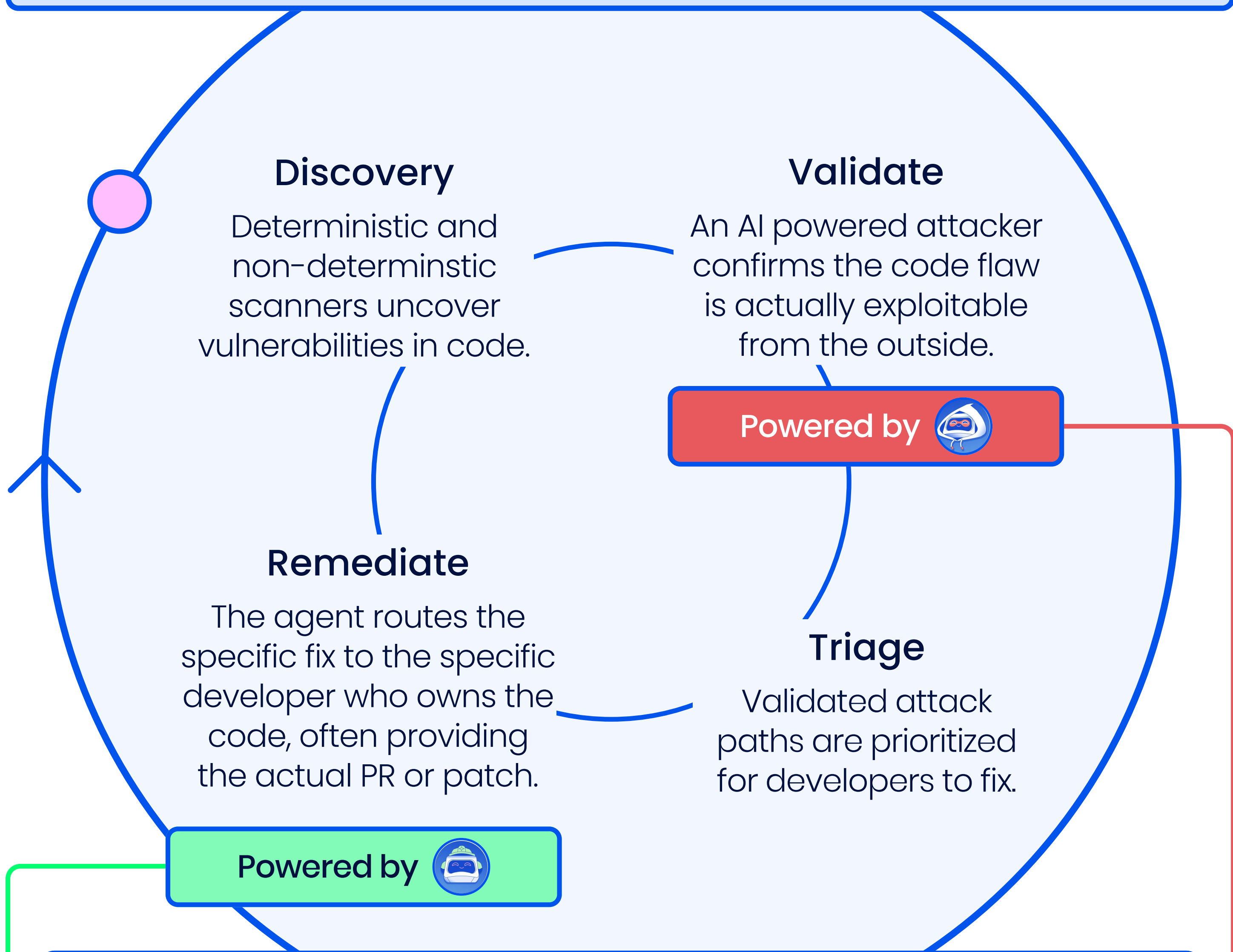
Use code-to-cloud context to generate tailored, root-cause fixes, assign ownership down to the individual developer, and route pull requests straight to code owners and their AI coding agents.



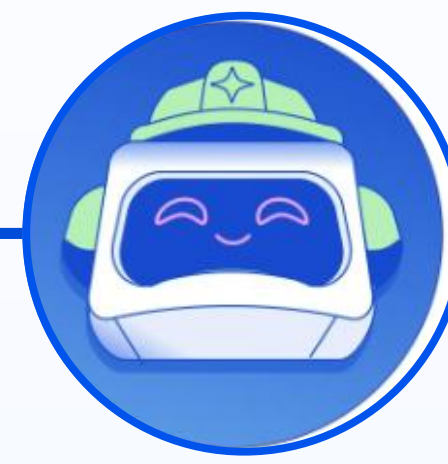
Govern

Track findings that aren't immediately fixed as security debt, with clear ownership and remediation SLAs, and report risk reduction over time.

The Agentic Application Security Flywheel



Operationalizing Application Security with Wiz Agents



Green Agent

Fix vulnerabilities as fast as you find them. The Green Agent uses the full context of your code, cloud, and runtime environment to orchestrate a tailored remediation plan starting with ownership mapping to fixes in code.



Red Agent

Go beyond theoretical code reachability to exploitability. The Red Agent acts as an AI powered attacker to validate whether code flaws can actually be exploited by a threat actor.

Actionable Steps for Teams

Ask These Four Questions (and act on the answers)

- 1 Do we have a risk-based strategy for deploying AI code analysis?
- 2 Do we use a mix of both automated, rules-based checks and deeper, AI-powered analysis to catch different types of security flaws?
- 3 Can we distinguish between a vulnerable finding and an exploitable attack path?
- 4 How mature are our remediation workflows in closing the gap between discovery and fix?

In the frontier AI era, finding vulnerabilities will become a commodity. Validation and machine-speed remediation are what separate resilient organizations from the rest.

