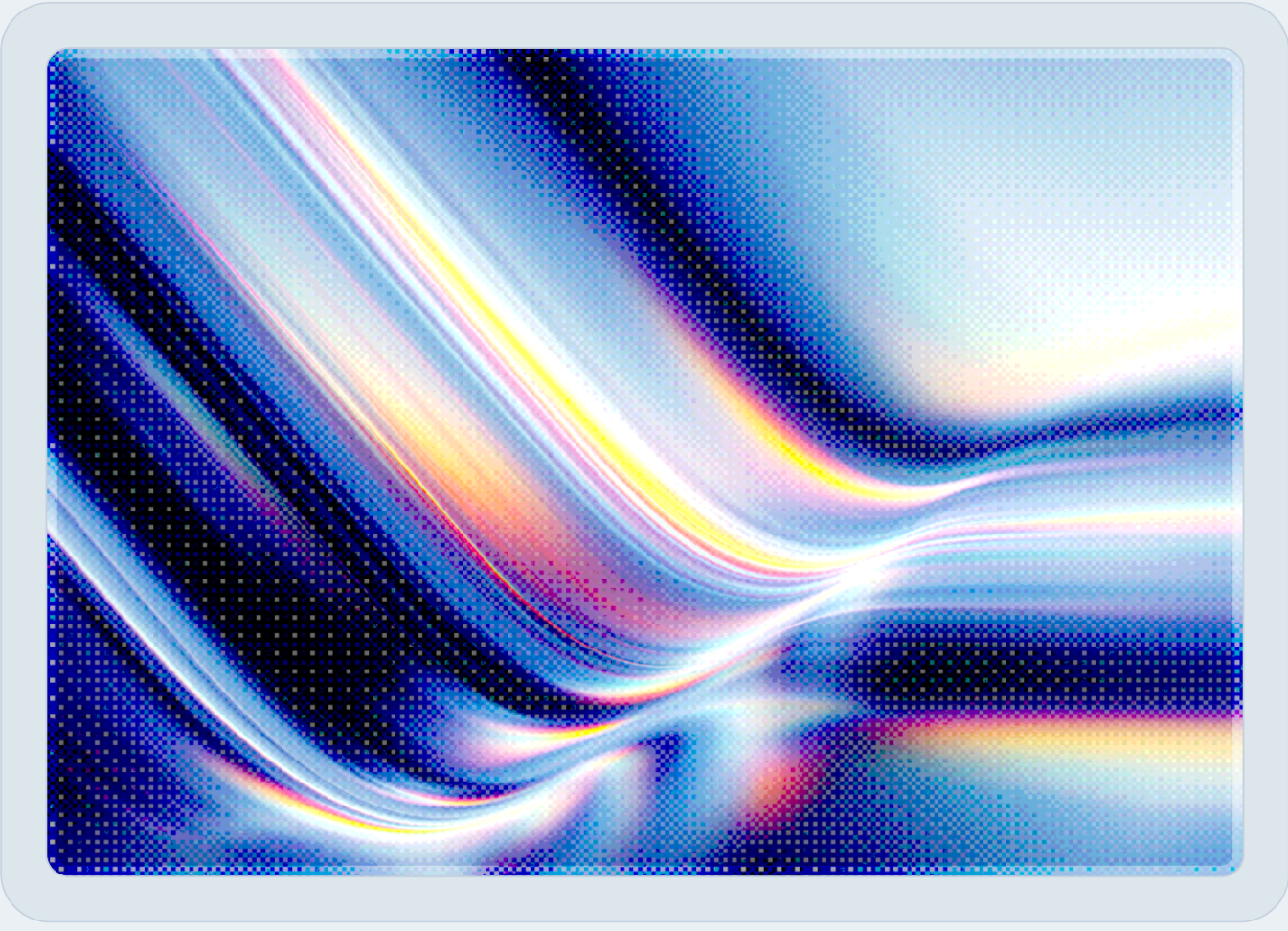
CSAI Foundation | Cloud Security Alliance

Who Owns the Bad Patch? Agentic Auto-Remediation Accountability

Security Risks of Agentic Auto-Remediation at Scale

2026-11-15

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

Agentic auto-remediation, in which an AI agent takes a security finding, writes a fix, validates it, and opens a pull request, began reaching general availability in 2026, led by GitHub. GitHub made agentic autofix for code scanning alerts generally available to GitHub Enterprise Cloud customers on October 15, 2026, letting teams select up to 25 alerts for the Copilot coding agent to fix together in a single pull request [1]. Snyk opened a preview of a remediation agent that runs scheduled dependency-upgrade campaigns across repositories [3]. Legit Security announced agents that open pull requests "across every affected repo in parallel" [4]. Taken together, vendor offerings now point toward remediation at fleet scale, with a human approving the result.

The evidence on patch correctness does not yet support treating a passing validation step as proof that a vulnerability is fixed. One 2026 study of 319 LLM-generated security patches found that 24.8% were fully correct, and that none of the patches for input-validation flaws in its sample fixed the vulnerability [8]. A separate study of seven agent frameworks confirmed 170 "silent failures": patches that passed functional checks but kept or introduced vulnerabilities. The LLM reviewer roles built into those frameworks also failed to catch them [9]. Both studies used research benchmarks and specific models rather than the commercial products above, so the figures indicate a pattern rather than product failure rates.

Accountability for a bad patch is spread across the scanner vendor, the agent vendor, the platform, the AppSec team that launched the campaign, and the engineer who clicked "approve." In practice it concentrates on the deploying organization, which is the pattern CSA's prior liability research describes for agentic systems generally [16]. Organizations should decide who owns an AI-authored fix before they scale these programs, not after the first regression or incomplete fix reaches production.

Remediation agents are also a new privileged identity. They typically require write access to many repositories and read untrusted content such as code, comments, and issues. Prior prompt-injection research against CI agents and repository-connected agents shows how that combination can be abused [11][12].

Background

Automated fix suggestions are not new. Dependency bots have opened version-bump pull requests for years, and GitHub introduced LLM-generated "Copilot Autofix" suggestions for CodeQL alerts in 2024. What changed in 2026 is the degree of agency, beginning with GitHub's July 2026 public preview of agentic autofix [2]. Earlier autofix systems proposed a diff inside an alert or pull request comment, and a developer applied it. Current systems operate as agents. GitHub's generally available feature explores the codebase, proposes a fix, re-runs CodeQL to check it, and opens a draft pull request with a summary of its reasoning and validation steps [1]. Snyk's remediation agent plans dependency upgrades, applies them in a hosted sandbox, checks builds, tests, and rescans, and then opens a pull request. AppSec teams define campaign scope, target repositories, and schedules, and campaigns can recur [3]. Legit's agents prioritize by reachability and exploitability, generate fixes, run tests, and open pull requests in parallel across affected services [4]. Semgrep's autofix, in public beta during 2026, pairs an LLM with deterministic analysis [5].

Research systems point in the same direction with more autonomy. In DARPA's AI Cyber Challenge final, competing cyber reasoning systems found 54 of 63 synthetic vulnerabilities and patched 43 of them across more than 54 million lines of code. They also identified 18 real, previously unknown vulnerabilities [7]. Google DeepMind's CodeMender upstreamed 72 security fixes to open-source projects over roughly six months, and every patch was reviewed by a human researcher before submission [6]. These results suggest that machine-generated patches can be useful. AIxCC demonstrated fully autonomous patching under competition conditions, while CodeMender, operating on real open-source projects, kept a human reviewer between the agent and the codebase.

The commercial products whose documentation CSA reviewed also keep a human gate. Snyk states that its agent does not merge changes and that developers "retain control over final review and merging" [3]. GitHub's documentation states that users must always review suggestions from Copilot Autofix and edit changes as needed before accepting them [15]. The security question this note examines is whether that human gate still functions as a control when the agent produces fixes faster than people can examine them, and who answers for the outcome when it does not.

Security Analysis

Validation Confirms the Alert Is Closed, Not That the Flaw Is Fixed

Agentic remediation pipelines typically validate a fix by re-running the scanner that raised the alert, running available tests, and confirming the build succeeds [1][3][4]. These checks are necessary, but they answer a narrower question than reviewers may assume. A rescan confirms that the specific pattern the analyzer flagged is no longer present. It does not confirm that the underlying weakness is gone, and it cannot detect a new weakness the analyzer has no rule for. Functional tests confirm that existing behavior is preserved, but functional test suites typically do not include a proof-of-vulnerability test for the flaw being fixed.

Empirical work in 2026 measured this gap directly. Al-Maamari evaluated 319 LLM-generated patches for 64 Java vulnerabilities on three axes: compilation, security (using proof-of-vulnerability tests), and functionality. Only 24.8% of patches were fully correct, and 51.4% failed on both security and functionality [8]. The study's Security Repair Score showed that models preserved functionality well (mean 0.832) but addressed the security flaw poorly (mean 0.251). Fix rates ranged from 0% for input-validation flaws to 45% for infinite-loop flaws [8]. Bai et al. analyzed 1,030 execution traces from seven agent frameworks and confirmed 170 silent failures. Of these, 48.2% omitted the needed fix, 30.6% introduced a new vulnerability, and 21.2% applied an inadequate fix. Because these patches passed the tests by definition, the more telling result is that the frameworks' LLM reviewer roles did not catch them either [9]. Both studies used research benchmarks and specific models (the Bai et al. study used a single model, GPT-4o-mini) rather than the commercial products described above, so their rates should not be read as product failure rates. The pattern they describe still matters for any pipeline whose validation relies on tests and rescans.

Scale Can Turn Individual Errors into Correlated Failures

Human-written fixes for the same vulnerability class across a hundred repositories vary, because different engineers make different choices. A remediation campaign run by one agent configuration produces fixes that tend to look alike. When the agent's approach is sound, consistency is an advantage. When it is subtly wrong, the same flaw is copied across the fleet. Bai et al. observed that similar insecure solutions recurred across different agent frameworks [9]. This suggests the failure modes come partly from shared model behavior rather than from any one tool. CSA's analysis therefore suggests that organizations should treat a remediation campaign more like a single change deployed many times than like many independent changes, and should apply the release-management discipline that implies.

Dependency-upgrade campaigns add a supply chain dimension. An agent that bumps a library across an estate is choosing which upstream version every service will trust next. Snyk notes that its hosted sandboxes use public package registries [3]. That is reasonable, but it means the agent's fix inherits whatever risk the newly selected version carries, including the possibility that the version itself is compromised. CSA has previously described AI coding agents as "an unaudited supply chain node" whose choices of dependencies and build actions receive less scrutiny than human contributors [17]. Remediation agents may be a particularly trusted form of that node, because their pull requests arrive labeled as security fixes.

The Human Approver Risks Becoming a Nominal Control

The products whose documentation CSA reviewed, GitHub and Snyk, leave the merge decision with a human, and their materials position that review as the primary safeguard [3][15]. The conditions agentic remediation creates, however, can push that safeguard toward a formality. A single GitHub autofix pull request can address up to 25 alerts [1], and campaign tools are explicitly designed to open many pull requests in parallel [3][4]. The pull request arrives with the agent's own explanation of why the fix is correct and a record of passing validation. Reviewers are therefore asked to check work that is pre-labeled as verified, in volumes set by the campaign rather than by their capacity.

Research on agentic development suggests review is already thin. Duma et al. studied AI-generated pull requests on GitHub in general, not security remediation pull requests specifically, and found that most received no review at all. When they were reviewed, the reviewers were largely other AI agents, with humans steering the agents rather than evaluating the code directly [10]. The authors caution that standard review metrics may overstate how much human oversight such pull requests actually receive [10]. If a remediation program counts "approved by a code owner" as evidence of human verification, it may be recording a signature rather than a security review. CSA's research on GhostCommit showed a related weakness from the attacker's side: content injected into a repository can defeat AI code review of the changes an agent submits [22].

The Remediation Agent Is a High-Value Target

To fix code across an organization, a remediation agent needs read and write access to many repositories, the ability to run builds, and it may require access to CI secrets. It also reads content an attacker may influence: source files, comments, issue text, dependency metadata, and alert descriptions. Aikido Security's PromptPwnd research showed that AI agents in GitHub Actions and GitLab pipelines could be steered by text in issues, pull requests, or commit messages into leaking secrets or changing repository data. The researchers reported at least five Fortune 500 companies were affected [11]. Invariant Labs showed that a malicious public GitHub issue could hijack an agent connected through the

GitHub MCP server and coerce it into leaking private repository data. They described the problem as architectural rather than a bug fixable on the server side [12]. CSA's May 2026 research note on prompt injection in AI-powered GitHub Actions documented the same class of risk in CI workflows [18].

A remediation agent combines the elements these attacks exploit: privileged write access, exposure to attacker-influenced content, and outputs that humans are inclined to trust. The scenario that follows is CSA's inference; no in-the-wild case has been publicly reported as of this writing. An attacker who can plant instructions in a code comment or vendored file that the agent reads while fixing a nearby alert could ask it to weaken a check, add a dependency, or change a workflow file. The change would be packaged inside a pull request titled as a security fix.

Accountability Is Distributed but Liability Is Not

When an AI-authored fix is incomplete or introduces a regression, several parties have contributed, but standard contracts and emerging regulation do not assign responsibility evenly. Table 1 summarizes the typical positions. The accountability column reflects CSA's assessment of common contractual and organizational patterns rather than a survey of specific agreements.

Table 1. Who contributes to an agentic fix, and where accountability tends to land

Party	What it controls	Typical accountability position (CSA assessment)
Scanner vendor	Detection rules, alert content, rescan logic used for validation	Alert accuracy; rarely responsible for fix correctness
Agent vendor / platform	Model, prompts, sandbox, validation workflow	Usually disclaims fitness of outputs; human review positioned as the control [3][15]
AppSec campaign owner	Scope, target repositories, schedule, risk thresholds	Owns program design; often not the approver of individual changes
Approving reviewer / code owner	Merge decision on each pull request	Formally approves, but may lack time or context to verify security semantics
Deploying organization	Production software and its customers	Bears regulatory and legal exposure for shipped product

The final row is where exposure concentrates. CSA's research on AI liability found that deploying an agent "does not transfer liability to the agent; it concentrates accountability on the deploying organization." It also noted that vendor contracts commonly cap liability and disclaim accuracy [16]. For organizations that are manufacturers under the EU Cyber Resilience Act, regulation reinforces this. Since September 11, 2026, they must submit an early warning within 24 hours of becoming aware of an actively exploited vulnerability, a vulnerability notification within 72 hours, and a final report no later than 14 days after a corrective measure is available [13][14]. Many organizations running remediation agents on internal code or pure SaaS offerings are not CRA manufacturers for that code, but where the obligation applies the stakes are direct. If the corrective measure is an AI-generated patch that closes the alert but not the flaw, the organization may believe it has remediated, and reported as much, while remaining exposed. U.S. proposals tracked by CSA in November 2026 would similarly place liability on operators and developers of autonomous agents, though none had become law as of this writing [19].

Recommendations

Immediate Actions

Organizations running or piloting agentic remediation should first assign an accountable owner to each campaign, distinct from the individual code owners who approve pull requests. That owner answers for the campaign's aggregate outcome, including correlated failures. Second, they should inventory the identities their remediation agents use and reduce them to the minimum repository scope and permissions each campaign needs. Agents should not hold organization-wide write tokens or deployment secrets. Third, they should label agent-authored security fixes clearly in commit metadata and pull request templates. Reviewers then know which changes came from an agent, and audit records can tell agent authorship apart from human approval.

Teams should also stop treating "rescan clean" as equivalent to "fixed" in their metrics and in any regulatory reporting. Where a vulnerability is known to be exploited, the corrective measure cited in reports should be verified independently of the agent's own validation.

Short-Term Mitigations

Over the next three to six months, organizations should strengthen validation so it tests the vulnerability rather than the alert. Practical steps include requiring a proof-of-vulnerability or regression test for high-severity fixes, running a second analyzer with different rules alongside the original scanner, and routing input-validation fixes, the class with the weakest results in published research [8], and other high-

consequence classes such as authorization, to specialist human review. Batch size deserves explicit limits. A pull request that bundles many alerts across unrelated code paths is harder to review well than several smaller ones.

Campaigns should roll out in stages. A fix pattern should be tried on a small set of repositories and observed before it reaches the full estate, the same way a configuration change would be. Agents should run in isolated sandboxes that keep secrets out of the build environment. Untrusted repository content such as issues, comments, and third-party files should be treated as potential injection input, following the controls CSA has described for AI agents in CI pipelines [18]. Dependency-upgrade campaigns should resolve packages through an internal proxy or allowlist instead of directly from public registries.

Strategic Considerations

Over the longer term, organizations should settle the allocation question in contracts and in policy. Procurement of remediation agents should address what the vendor warrants about validation, how failures are reported, and whether audit logs of the agent's reasoning and tool calls are retained and available to the customer. Internally, review policy should state what an approving reviewer is attesting to when approving an agent-authored security fix, and should provide time and tooling that make that attestation realistic.

Organizations should also measure outcomes rather than activity. Merge rates and alerts closed are common headline figures in vendor reporting, but they do not show whether fixes worked. More useful measures include the rate of reopened or recurring findings, the rate of post-merge regressions traced to agent fixes, and the results of periodic expert audits of a sample of merged fixes. CSA anticipates that these figures, rather than throughput, will determine whether agentic remediation reduces risk or relocates it.

CSA Resource Alignment

This note builds on several CSA publications that address agentic development and accountability. CSA's *AI Liability Inflection: Enterprise Accountability in the Agentic Era* (June 2026) provides the legal and contractual foundation for the accountability analysis above. It argues that deploying organizations retain liability under an emerging "reasonable oversight" standard unless they can show that monitoring and safety controls were operating when the agent acted [16]. Agentic remediation is a concrete case of that principle. The campaign owner role, retained audit trails, and independent validation recommended here are the kinds of documented oversight that paper describes.

AI Coding Agents: An Unaudited Supply Chain Node (July 2026) argues that coding agents have become supply chain dependencies that organizations are not yet inventorying, signing, or auditing, and that provenance must extend to the generating model, agent framework, and context [17]. Remediation agents are a high-trust instance of this node, and that paper's provenance recommendations apply directly to agent-authored security fixes. CSA's *Prompt Injection in AI-Powered GitHub Actions* (May 2026) details the CI-level attack class and defensive controls that this note recommends for remediation agents operating with repository write access [18]. *GhostCommit: Image-Based Prompt Injection Defeats AI Code Review* shows how injected repository content can undermine the review step on which agentic remediation programs depend [22].

For control mapping, the CSA AI Controls Matrix (AICM) v1.1 provides control objectives that cover the issues raised here. These include vulnerability remediation in Threat and Vulnerability Management (TVM), secure development in Application and Interface Security (AIS), change management in Change Control and Configuration Management (CCC), and agent credential controls in Identity and Access Management (IAM) [20]. Teams threat-modeling a remediation pipeline can use CSA's MAESTRO framework to analyze the agent, its tools, and its interactions with CI and repository systems layer by layer [21].

References

[1] GitHub. "Agentic autofix for code scanning alerts general availability." GitHub Changelog, October 15, 2026.

[2] GitHub. "Agentic autofix for code scanning alerts in public preview." GitHub Changelog, July 10, 2026.

[3] Snyk. "Evo Remediation Agent Open Preview." Snyk Blog, October 22, 2026.

[4] Legit Security. "Legit Security Launches Autonomous Remediation Agents to Close the Gap Between AI-Led Attacks and AI-First Development." Legit Security, June 2026.

[5] Semgrep. "Accelerate and Automate Remediation with Semgrep Autofix." Semgrep Blog, March 16, 2026.

[6] Google DeepMind. "Introducing CodeMender: an AI agent for code security." Google DeepMind Blog, October 2025.

[7] DARPA. "AI Cyber Challenge marks pivotal inflection point for cyber defense." DARPA, August 8, 2025.

[8] Al-Maamari, A. "Why LLMs Fail: A Failure Analysis and Partial Success Measurement for Automated Security Patch Generation." arXiv:2603.10072, March 2026.

[9] Bai, W., Waseem, M., Rasheed, Z., Peltonen, J., and Abrahamsson, P. "When Passing Tests Hides Vulnerabilities: An Empirical Study of Silent Failures in Agentic Systems." arXiv:2609.10548, June 2026.

[10] Duma, K., Wróblewski, P., Bobińska, J., Winiarska, J., and Przymus, P. "These Aren't the Reviews You're Looking For: How Humans Review AI-Generated Pull Requests." arXiv:2605.02273 (EASE 2026), May 2026.

[11] Aikido Security. "PromptPwnd: Prompt Injection Vulnerabilities in GitHub Actions Using AI Agents." Aikido Blog, December 4, 2025 (updated March 17, 2026).

[12] Invariant Labs. "GitHub MCP Exploited: Accessing private repositories via MCP." Invariant Labs Blog, May 2025.

[13] European Commission. "Cyber Resilience Act – Reporting obligations." Shaping Europe's Digital Future, accessed November 2026.

[14] Jones Day. "EU Cyber Resilience Act: 24-Hour Reporting Duties Start September 11, 2026." Jones Day Insights, July 2026.

[15] GitHub. "Application card: GitHub security and quality AI features." GitHub Docs, accessed November 2026.

[16] Cloud Security Alliance. "AI Liability Inflection: Enterprise Accountability in the Agentic Era." CSA AI Safety Initiative, June 27, 2026.

[17] Cloud Security Alliance. "AI Coding Agents: An Unaudited Supply Chain Node." CSA AI Safety Initiative, July 8, 2026.

[18] Cloud Security Alliance. "Prompt Injection in AI-Powered GitHub Actions." CSA AI Safety Initiative, May 3, 2026.

[19] Cloud Security Alliance. "Liability Reaches the Operator: Agent Accountability, CLAIM, and 48-Hour Reporting." CSA AI Safety Initiative, November 3, 2026.

[20] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2025.

[21] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 6, 2025.

[22] Cloud Security Alliance. "GhostCommit: Image-Based Prompt Injection Defeats AI Code Review." CSA AI Safety Initiative, 2026.