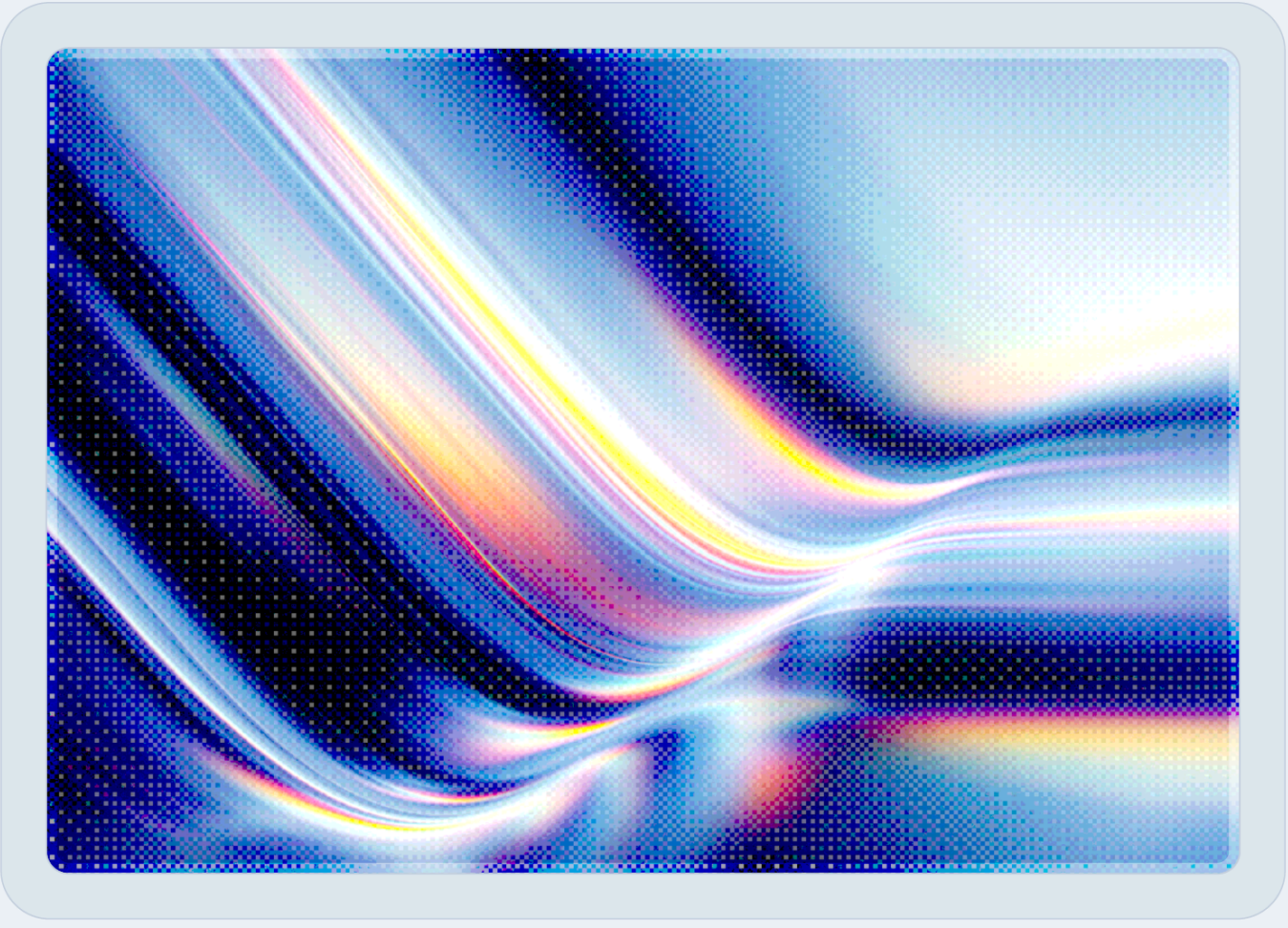
CSAI Foundation | Cloud Security Alliance

Amazon Q's MCP Flaw: When AI Assistants Become Attack Paths

CVE-2026-12957 and a Recurring Trust Gap in AI Coding Assistants

2026-07-07

AI-assisted Rapid Research



CSAI

cloud security alliance®

Key Takeaways

- Wiz Research disclosed a high-severity flaw (CVE-2026-12957) — assigned a CVSS score of 8.5 [3] — in the Amazon Q Developer extension for Visual Studio Code that let a malicious repository silently execute code and steal live AWS credentials the moment a developer opened the project folder [1][2].
- In CSA's assessment, the root cause was structural rather than incidental: the extension auto-loaded Model Context Protocol (MCP) server definitions from a workspace file, `.amazonq/mcp.json`, without any consent prompt, and the resulting child processes inherited the developer's entire environment, including AWS access keys and SSH agent sockets [1][3].
- A related flaw, CVE-2026-12958, involved insufficient symlink validation and was patched in a later release [1][4].
- Amazon Web Services fixed the primary flaw, CVE-2026-12957, in Language Servers for AWS version 1.65.0 on May 12, 2026; full remediation of the related symlink issue followed in version 1.69.0, alongside minimum plugin version bumps for the VS Code, JetBrains, Eclipse, and Visual Studio toolkits, and most environments update automatically [1][4].
- This is not an isolated incident. Comparable MCP trust bypasses have surfaced in Anthropic's Claude Code (CVE-2025-59536), Cursor (CVE-2025-54136, "MCPoison"), and Windsurf (CVE-2026-30615), indicating a systemic design pattern across the AI coding assistant category rather than a defect specific to one vendor [3][5][6][7].
- Enterprises that have adopted AI coding assistants should treat workspace-level MCP configuration as an untrusted, executable artifact — equivalent to a build script committed by a stranger — and govern it accordingly.

Background

Amazon Q Developer is AWS's AI-powered coding assistant, distributed as an extension for Visual Studio Code, JetBrains IDEs, Eclipse, and other editors, and designed to generate code, answer questions about a codebase, and automate development tasks directly inside a developer's working environment. Amazon Q, like many AI coding assistants that have emerged since 2024, supports the Model Context

Protocol, an open specification that Anthropic introduced to let AI applications discover and call external tools — databases, build systems, internal APIs — through a standardized server interface. MCP has been adopted rapidly across the AI assistant ecosystem, likely in part because it lets a coding assistant extend its own capabilities on the fly, pulling in whatever tools a given project needs without requiring the assistant vendor to hard-code every possible integration.

That extensibility comes from a specific mechanic: many MCP-enabled tools look for a configuration file inside the project a developer opens — in Amazon Q's case, `.amazonq/mcp.json` — and, if present, treat it as instructions for which MCP servers to start and how. Because that file travels with the repository itself, whoever controls the repository content effectively controls what the AI assistant will run on the machine of anyone who opens it. This is a meaningful departure from how developer tooling has traditionally handled untrusted code. Editors and IDEs have spent the better part of a decade building "workspace trust" models — Visual Studio Code, for instance, prompts users before running tasks, debuggers, or extensions defined by an unfamiliar folder — specifically because opening a repository has always carried some risk, and because that risk needs to be gated by explicit user consent rather than assumed away.

The Wiz Research team, led by researcher Maor Dokhanian, found that the Amazon Q Developer extension did not carry that consent gate forward into its MCP implementation. According to Wiz, "the extension loaded `.amazonq/mcp.json` from the workspace root immediately upon opening the folder. No dialog asked the user to approve these MCP servers" [1]. In practical terms, an AI assistant that developers have come to trust as a productivity aid was, in this configuration, willing to execute arbitrary commands supplied by whoever authored the project folder — including an attacker who had never interacted with the victim beyond publishing a repository or compromising a dependency the victim pulled in. Security researchers have separately warned that this exposure extends to superficially innocuous contexts, such as a job-application coding test in which a candidate is asked to clone and open an employer-supplied repository.

Security Analysis

The attack chain Wiz demonstrated required only two steps from the victim's perspective: cloning a repository and opening it in VS Code with the Amazon Q extension installed. From there, the exploit unfolded automatically. The extension read `.amazonq/mcp.json`, found an entry describing an MCP server, and launched the command it specified — in Wiz's proof of concept, a shell command disguised as an innocuous "build helper" that ran `aws sts get-caller-identity` and posted the result to an attacker-controlled endpoint [1]. Because that command ran as a normal child process of the developer's own IDE session, it inherited the full ambient environment: active AWS access keys,

session tokens, cloud CLI credentials, and any SSH agent sockets available to the user. Wiz's test confirmed that this was sufficient to capture a live, usable AWS session belonging to the victim, at which point an attacker could pivot from a single developer laptop into whatever cloud infrastructure that developer's credentials could reach — reading data, provisioning resources, or establishing longer-term persistence through a backdoored IAM identity [1][2].

Two design failures compounded to make this possible, and it is worth separating them because each represents a distinct control that failed. First, the extension treated the mere presence of a workspace configuration file as sufficient authorization to start a server and run its associated command — there was no analog to the workspace-trust or "do you want to allow this?" prompt that has become standard practice for less privileged IDE features. Second, even setting consent aside, the servers that did run were not sandboxed or scoped to a minimal environment; they inherited everything the parent VS Code process could see. Either control alone would likely have prevented the credential theft outcome: a consent prompt would have given an attentive developer a chance to reject an unfamiliar server, and environment isolation would have denied a rogue process access to ambient secrets even if it did run. The absence of both, together, converted a convenience feature into what Wiz characterized as a path "from git clone to cloud compromise" [1].

AWS Security acknowledged Wiz's report the same day it was submitted, April 20, 2026, and deployed a fix for the primary flaw, CVE-2026-12957, through a Language Server update to version 1.65.0 on May 12, 2026 — roughly three weeks later [1][4]. The Common Vulnerabilities and Exposures identifier CVE-2026-12957 was formally assigned on June 23, 2026, alongside a related identifier, CVE-2026-12958, covering a separate but adjacent flaw involving insufficient symlink checks in the same MCP-loading code path; full remediation of that second issue followed in a later release, Language Servers for AWS version 1.69.0 [1][4]. Wiz published its public writeup on June 26, 2026, consistent with standard coordinated-disclosure practice of withholding technical detail until a fix has had time to propagate [1][3]. The remediation, complete as of version 1.69.0, introduces an explicit consent prompt before any MCP server defined in a workspace file is allowed to run — giving developers visibility into, and the ability to reject, commands they did not author — and carries corresponding minimum-version requirements for the VS Code (2.20+), JetBrains (4.3+), Eclipse (2.7.4+), and Visual Studio (1.94.0.0+) plugins [3][4]. AWS has stated that the underlying Language Server updates automatically in most network configurations, meaning affected users are protected once they reload their IDE, though organizations with restrictive network policies or pinned extension versions should verify their update status directly rather than assume automatic remediation applies to them.

What elevates this beyond a single-vendor patch note is how closely it mirrors flaws independently identified in competing AI coding assistants over the preceding year. Check Point Research disclosed "MCPoison" (CVE-2025-54136) in Cursor in mid-2025, in which trust granted to an MCP configuration entry at first approval was never re-validated when the entry's underlying command was later swapped

out — letting an attacker poison an already-trusted configuration after the fact [5]. Separately, Check Point identified CVE-2025-59536 in Anthropic's Claude Code, where project-level hooks and MCP server settings could execute before any trust dialog was rendered, again allowing arbitrary command execution and credential exfiltration from a freshly opened, untrusted project [6]. Most recently, researchers documented CVE-2026-30615 in Windsurf, a zero-click variant in which attacker-controlled HTML content processed by the assistant could silently rewrite the local MCP configuration and register a malicious server with no user interaction required at all — a step beyond the other cases, which still required a developer to open a project or approve an initial prompt [7]. Taken together, these four disclosures span four different vendors and four different specific coding defects, but they share a single underlying pattern: MCP configuration that travels with project content is treated with more trust than the content itself warrants, and the servers that configuration launches run with more ambient privilege than their function requires. Separate ecosystem-wide research from OX Security into MCP STDIO command injection reached a similar conclusion, finding related trust and execution weaknesses recurring across multiple AI development tools rather than being confined to any single product [10].

Recommendations

Immediate Actions

Security teams should confirm that every developer machine running Amazon Q Developer has updated to Language Servers for AWS version 1.69.0 or later, along with the corresponding minimum plugin version for whichever IDE integration is in use, and should not assume automatic updates have reached every endpoint, particularly on networks with restricted outbound connectivity or centrally managed extension deployments. Where feasible, teams should audit recent developer activity for evidence of the exploit pattern — unexpected `.amazonq/` directories in cloned repositories, unfamiliar MCP server entries in workspace configuration files, or anomalous outbound calls to `aws sts get-caller-identity` or credential-adjacent AWS API calls immediately following a repository clone. Any AWS credentials that were active on a developer machine that opened an untrusted repository before the May 12, 2026 patch should be treated as potentially exposed and rotated as a precaution.

Short-Term Mitigations

Beyond the immediate patch, organizations should extend the same scrutiny to every other AI coding assistant in active use, since the underlying pattern — not just this specific CVE — is what created the risk. That means checking whether Cursor, Claude Code, Windsurf, or any other MCP-enabled tool deployed

in the environment has applied its respective fix, and, more durably, establishing a policy that developer machines should not hold long-lived, broadly scoped cloud credentials in the ambient environment where an IDE or its extensions can reach them. Short-lived, narrowly scoped credentials issued through a broker or federated identity provider limit the blast radius of any single compromised process, regardless of which tool that process came from. Security and platform engineering teams should also inventory which AI coding assistants are approved for use with production-adjacent credentials, since the risk profile of an assistant that silently executes workspace-defined commands is materially different from one that requires explicit approval for every action.

Strategic Considerations

The recurrence of this same failure mode across four unrelated vendors suggests that consent and privilege boundaries for AI coding assistants deserve treatment as a first-class security requirement during tool evaluation and procurement, not as an assumed property of any product that markets itself as enterprise-ready. Organizations building internal standards for AI development tooling should require, at minimum, that any tool capable of executing project-defined configuration present an explicit, human-reviewable approval step before doing so, and that any process the tool spawns run with a deliberately scoped environment rather than full inheritance of the parent session's credentials and secrets. As MCP and similar extensibility protocols become the default mechanism by which AI assistants gain capability, the security community's task is less about patching individual implementations after the fact and more about establishing what a trustworthy default posture looks like before the next vendor repeats the same design decision.

CSA Resource Alignment

The trust-boundary failure at the center of this incident — an autonomous component acting on unauthenticated, externally supplied instructions with excessive inherited privilege — falls within the class of risk CSA's Agentic AI Threat Modeling Framework, MAESTRO [8] models. MAESTRO's layered approach explicitly calls out the absence of clear trust boundaries between an agent and the tools or configuration it consumes as a distinct category of agentic risk, separate from traditional software vulnerabilities; the Amazon Q flaw is a concrete instance of that category, where a workspace file authored by an untrusted third party was treated as an implicitly trusted instruction set by the assistant's tool-invocation layer. Teams applying MAESTRO to their own AI-assisted development pipelines should specifically model the MCP configuration-loading step as an external trust boundary requiring explicit mediation, not an internal implementation detail.

The credential-inheritance dimension of this incident — where a spawned process gained unrestricted access to the developer's ambient AWS session — maps to the identity and access management guidance in CSA's AI Controls Matrix (AICM) v1.1 [9]. AICM's Identity and Access Management (IAM) domain establishes the expectation that AI system components, including agents and their invoked tools, operate under least-privilege, scoped credentials rather than inheriting the full privilege of their host process or user session; this incident is a direct illustration of what happens when that control is absent from a widely deployed developer tool. Organizations conducting AICM-based assessments of their AI development toolchain should treat "does this tool's spawned processes inherit ambient credentials" as a specific control question within the IAM domain, since the answer was affirmatively unsafe across all four vendors discussed above.

References

[1] Dokhanian, M. "MCP Auto-Execution: From Git Clone to Cloud Compromise in Amazon Q VS Code Extension." Wiz Research, June 26, 2026.

[2] Cybersecurity News. "Amazon Q Vulnerability Let Attackers Execute Code and Access Sensitive Cloud Environments." Cybersecurity News, June 2026.

[3] The Hacker News. "Amazon Q Developer Flaw Could Let Malicious Repos Run Code via MCP Configs." The Hacker News, June 26, 2026.

[4] Amazon Web Services. "AWS Security Bulletin 2026-047-AWS." AWS Security Bulletin, June 23, 2026.

[5] Check Point Research. "Cursor IDE's MCP Vulnerability – MCPoison." Check Point Research, 2025.

[6] Check Point Research. "Caught in the Hook: RCE and API Token Exfiltration Through Claude Code Project Files." Check Point Research, 2026.

[7] SentinelOne. "CVE-2026-30615: Windsurf Prompt Injection RCE Vulnerability." SentinelOne Vulnerability Database, 2026.

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

[9] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." CSA, June 22, 2026.

[10] OX Security. "MCP Supply Chain Advisory: RCE Vulnerabilities Across the AI Ecosystem." OX Security, April 15, 2026.