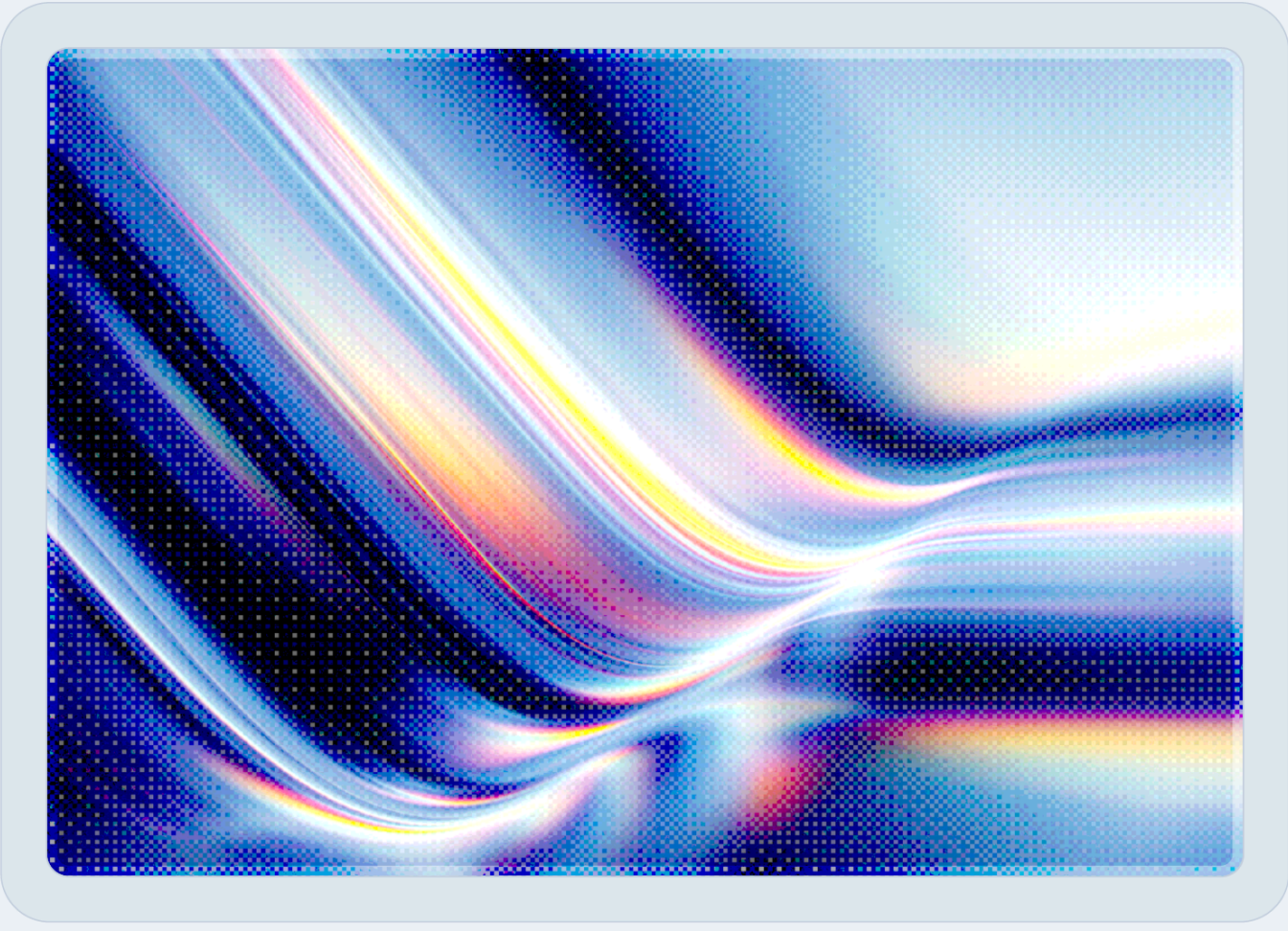
CSAI Foundation | Cloud Security Alliance

MCP Auto-Execution: AI Coding Assistants and Credential Theft

When a Git Clone Becomes a Cloud Compromise

2026-06-29

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Multiple widely-used AI coding assistants — including Amazon Q Developer and Anthropic's Claude Code — contained vulnerabilities that allowed a malicious git repository to execute arbitrary commands and steal cloud credentials without user interaction beyond opening the project.
- The attack surface stems from a systemic design pattern: MCP (Model Context Protocol) server configurations embedded in repository files are automatically initialized at project open in affected versions, before trust verification is performed.
- Two high-severity CVEs in Claude Code (CVE-2025-59536, CVSS 8.7, High; CVE-2026-21852, CVSS 5.3, Medium) and two in Amazon Q Developer (CVE-2026-12957, CVSS 8.5, High; CVE-2026-12958, a companion vulnerability) have been patched, but the underlying architectural pattern — auto-executing repository-controlled configurations — remains a concern across the broader AI tooling ecosystem.
- GitGuardian's 2026 Secrets Sprawl report found 24,008 unique secrets exposed in MCP configuration files on public GitHub, with 8.8% confirmed still valid at time of scan [1].
- Organizations should treat repository-embedded MCP configurations as untrusted code requiring explicit approval workflows and rotate credentials immediately after any exposure to AI coding tools.

Background

The Model Context Protocol (MCP) was introduced by Anthropic in late 2024 as an open standard enabling AI assistants to connect with external tools, APIs, databases, and development environments through a uniform interface [2]. By 2025, virtually every major AI coding assistant — Amazon Q Developer, Claude Code, GitHub Copilot extensions, Cursor, and others — had integrated MCP server support to enhance developer productivity. MCP servers run as local processes spawned by the AI assistant, providing capabilities like file system access, shell execution, API calls, and cloud service integrations. This expanded capability is also what makes MCP a significant attack surface — one that attackers have actively exploited, as the vulnerabilities documented below demonstrate.

AI coding assistants run with the developer's OS identity and inherited shell environment, giving them the same access to secrets as the developer themselves: AWS credentials stored in `~/.aws/credentials`, API keys set as environment variables, SSH agent sockets, and cloud CLI session tokens. An AI assistant that can be coerced into spawning a malicious process inherits all of this privilege immediately, without any additional exploitation step. This is not a theoretical concern — it is the mechanism behind the cluster of CVEs disclosed across multiple AI coding tools in the first half of 2026.

Security Analysis

The Core Vulnerability Pattern

The attack exploits a straightforward trust gap: AI coding assistants treat repository-embedded configuration files as instructions to follow, not as untrusted input to sanitize. When a developer clones a repository and opens it in their IDE, the AI assistant initializes MCP servers defined in workspace configuration files (`.amazonq/mcp.json` for Amazon Q, `.mcp.json` for Claude Code) as part of its startup sequence. This initialization runs before the user has reviewed the configuration file and, in the flawed versions, before any workspace trust verification dialog could be surfaced.

Because the spawned MCP server processes are children of the IDE extension, they inherit the full process environment. On a typical developer workstation this means inheriting `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`, `ANTHROPIC_API_KEY`, GitHub personal access tokens, and any other credentials the developer has loaded into their shell session. An attacker controlling the `.mcp.json` content can direct these processes to exfiltrate those values to external infrastructure before the developer is aware the repository is malicious.

CVE-2025-59536 and CVE-2026-21852: Claude Code

Check Point Research published findings in early 2026 detailing two distinct flaws in Anthropic's Claude Code that together allowed both arbitrary local code execution and API key exfiltration through repository-controlled files [3].

CVE-2025-59536 (CVSS 8.7, High) exploited Claude Code's hook execution mechanism. Repository settings files could define shell commands that Claude Code would execute automatically as part of its lifecycle — commands that ran before the workspace trust dialog was presented to the user. A developer

who cloned a poisoned repository and opened Claude Code would trigger arbitrary shell execution with no visible indication that anything unusual had occurred.

CVE-2026-21852 (CVSS 5.3, Medium) took a subtler approach: a single environment variable override in the repository configuration could silently redirect Claude Code's outbound API traffic — including the full `Authorization` header containing the developer's Anthropic API key — to attacker-controlled infrastructure. Because this interception occurred before the trust dialog appeared, the credential was captured during what the user perceived as normal startup. Both vulnerabilities have been patched in current Claude Code releases, and Anthropic recommends that any affected users rotate their API keys immediately.

CVE-2026-12957 and CVE-2026-12958: Amazon Q Developer

Wiz Research disclosed a parallel class of vulnerabilities in the Amazon Q Developer extension for Visual Studio Code in June 2026, following responsible disclosure to Amazon on April 20, 2026 [4]. CVE-2026-12957 (CVSS 8.5, High) allowed Amazon Q to automatically load and execute MCP server commands from `.amazonq/mcp.json` files within the opened workspace, without workspace trust checks or user consent. The absence of workspace isolation meant spawned processes immediately inherited AWS IAM credentials from the developer's environment — `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` — giving an attacker the same AWS access as the developer's identity can reach.

Amazon deployed an initial fix on May 12, 2026, and completed full remediation in Language Servers for AWS version 1.69.0 [5]. CVE-2026-12958, a companion vulnerability addressed in the same 1.69.0 release, reflected a related trust boundary failure in the same component; detailed CVSS scoring for this CVE was not publicly available at time of writing.

Delivery Vectors and Realistic Attack Scenarios

The common thread in all documented attacks is that the developer must open the malicious repository — there is no network-based remote exploitation. However, the social engineering surface for convincing a developer to clone a repository is broad. Typosquatted package names in npm, PyPI, or other registries can direct developers to malicious repositories — either through install scripts that reference them or through README links that lead a developer examining the package source to clone an attacker-controlled project. Malicious pull requests to popular open-source projects expose any contributor who checks out the PR branch locally.

Techniques resembling the North Korean-attributed "Contagious Interview" campaign have demonstrated that fake technical interview challenges — sent by recruiters on LinkedIn or other professional networks — are an effective pretext for inducing developers to clone and run attacker-controlled code. Mitiga documented an AI agent–specific variant of this approach in 2026, in which the technique was deployed against a developer working in an AI-enabled environment [6]. Even legitimate repositories are not inherently safe: a supply chain compromise of a trusted upstream project could introduce a malicious `.mcp.json` in a single commit that affects every downstream contributor who pulls the change.

Ecosystem-Wide Scope

These vulnerabilities are not isolated to a single product. Check Point and Wiz published independent research on overlapping auto-execution patterns in different AI coding tools in early-to-mid 2026 [3] [4], and separate research documented that Anthropic's official MCP SDKs for Python, TypeScript, Java, and Rust passed parameters from incoming STDIO transport configurations directly to the host OS shell without sanitization, creating a shell injection path in any application built on those SDKs before patching. A systematic review published on arXiv in January 2026 found that Claude Code, GitHub Copilot Agent, and Cursor were all susceptible to prompt injection attacks delivered through code comments, with attack success rates exceeding 85% against contemporary defenses [7][8].

Separately, research published by GitGuardian in their 2026 State of Secrets Sprawl report documented 24,008 unique secrets — including API keys, database passwords, and cloud credentials — present in MCP-related configuration files on public GitHub repositories, with 2,117 of those (8.8%) confirmed valid at scan time [1]. Nearly half of surveyed MCP server implementations recommended storing secrets in plaintext `.env` files or JSON configuration files as part of their official setup guidance, normalizing insecure practices at the protocol level [1].

Recommendations

Immediate Actions

Organizations with active development teams using AI coding assistants should take the following steps without delay. First, audit and update all AI coding assistant extensions to their latest patched versions — verify that Claude Code has been updated to address CVE-2025-59536 and CVE-2026-21852, and that Amazon Q Developer's language server is at version 1.69.0 or later. Second, any developer who ran

an affected version of these tools on repositories not fully controlled by their own organization should treat their Anthropic API keys, AWS credentials, and any other secrets present in the shell environment as potentially compromised and rotate them immediately. Third, organizations should inventory all MCP configuration files (`.mcp.json` , `.amazonq/mcp.json` , and equivalents for other tools) present in their internal repositories and verify that none contain externally-controlled MCP server endpoints.

Short-Term Mitigations

Development teams should establish explicit MCP governance policies that mirror the maturity already expected for other forms of executable code. Repository-embedded MCP configurations should be treated as code requiring review in the pull request process, with the same scrutiny applied to any shell command or CI/CD pipeline modification. AI coding assistant workspace trust settings should be configured to require explicit user approval before initializing any MCP server from a newly opened repository. Where possible, organizations should prefer organizational allow-lists for approved MCP servers over permitting arbitrary repository-defined configurations to execute.

Secrets management practices need to be hardened in development environments specifically because AI coding assistants now amplify the blast radius of environment-level credential exposure. AWS IAM roles for developers should use short-lived session tokens via SSO rather than long-lived access keys stored in credential files. API keys for AI services should be scoped to the minimum necessary permissions and issued per-developer or per-project rather than shared. Shell initialization files should avoid loading credentials into the default environment where possible, using tools like HashiCorp Vault, AWS Secrets Manager, or 1Password CLI for on-demand credential retrieval.

Strategic Considerations

At a strategic level, the pattern revealed by these vulnerabilities reflects a category of risk that will require sustained attention as the AI coding assistant ecosystem matures. The MCP specification itself does not mandate workspace trust verification before server initialization; individual tool vendors have implemented trust controls with varying degrees of rigor, and security review processes have not kept pace with the rate of new MCP-compatible tool releases — as evidenced by multiple research teams independently discovering similar vulnerabilities across different products in overlapping timeframes. Security teams should engage with AI tooling procurement decisions and establish standards for minimum trust controls before approving any AI coding assistant for use on development workstations with access to production credentials.

Organizations building internal tools on MCP SDKs should conduct their own security reviews of configuration handling, particularly around the STDIO transport, and ensure that none of their applications pass repository-supplied parameters to shell execution without sanitization. Incident response playbooks should be updated to include AI coding tool environments as a potential source of credential exposure, alongside the browser, email, and endpoint categories already covered. When investigating a credential compromise with no obvious source, the developer's AI coding assistant workspace configurations are now a relevant forensic artifact.

CSA Resource Alignment

This research note connects to several active Cloud Security Alliance frameworks and programs.

MAESTRO (Multi-Agent Environment, Security, Threat Risk, and Outcome) is CSA's agentic AI threat modeling framework [9]. This attack class maps most naturally to MAESTRO's Layer 3 (Agent Frameworks) and Layer 1 (Foundation Models) threat categories, specifically the tool interface abuse and supply chain compromise threat vectors. Security teams performing threat models on AI coding assistant deployments should use MAESTRO to enumerate the full privilege inheritance surface of AI agent processes.

AICM (AI Controls Matrix) provides the security control framework that organizations can use to assess and improve their posture against the risks described here. Control domains covering AI supply chain security (SC), identity and access management (IAM), and configuration management (CM) are directly applicable. Application providers integrating MCP-capable AI assistants should reference AICM's Application Provider implementation guidelines to understand their shared responsibility obligations.

CSA's Zero Trust guidance is relevant to the credential theft dimension of these attacks. A Zero Trust architecture that enforces continuous authentication and authorization — rather than relying on long-lived credentials stored in developer shell environments — can reduce the blast radius of an MCP auto-execution exploit by limiting the scope and validity window of captured credentials, though it does not prevent the initial exfiltration.

CSA's AI Safety Initiative has previously examined related attack patterns in AI Agent Prompt Injection: The New CI/CD Supply Chain Threat and MCP Security Crisis: Systemic Design Flaws in AI Agent Infrastructure, which provide additional technical context on prompt injection delivery mechanisms and MCP server exposure.

References

[1] GitGuardian. "The State of Secrets Sprawl 2026." GitGuardian Annual Report, 2026.

[2] Anthropic. "Introducing the Model Context Protocol." Anthropic, November 2024.

[3] Check Point Research. "Caught in the Hook: RCE and API Token Exfiltration Through Claude Code Project Files | CVE-2025-59536 | CVE-2026-21852." Check Point Research, February 2026.

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

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

[6] Mitiga. "The Take-Home Test: AI Agent–Driven Cloud Account Compromise Through Poisoned Coding Assessments." Mitiga Security, 2026.

[7] SecurityWeek. "Claude Code, Gemini CLI, GitHub Copilot Agents Vulnerable to Prompt Injection via Comments." SecurityWeek, April 2026.

[8] Maloyan, N. and Namiot, D. "Prompt Injection Attacks on Agentic Coding Assistants: A Systematic Analysis of Vulnerabilities in Skills, Tools, and Protocol Ecosystems." arXiv, January 2026.

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