INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-29 07:11 UTC

Amazon Q AI Coding Assistant Enabled Cloud Credential Theft via Malicious Repository Injection

SECURITY ANALYSIS | CRITICAL | CVSS 8.1

SCC Item ID	SCC-STY-2026-0297
Type	Security Analysis
Severity	CRITICAL
CVSS Base Score	8.1
Affected Products	Amazon Q Developer (AI coding assistant), patched version unspecified in available sources
Published	2026-06-27
Discovery Source	Gemini

Executive Summary

A now-patched vulnerability in Amazon Q Developer, AWS's AI-powered coding assistant, allowed attackers to embed malicious instructions inside code repositories that the AI agent would execute, potentially exfiltrating developer AWS credentials to attacker-controlled infrastructure. ReversingLabs characterized the incident as a near-miss supply chain event that could have exposed credentials for a large developer population. This attack pattern signals a maturing threat category: AI coding agents are becoming a viable attack surface for credential theft and supply chain compromise, and their trust model demands the same scrutiny organizations apply to third-party software.

Technical Analysis

The attack exploited a prompt injection vulnerability, a class of weakness specific to AI systems that process untrusted text as instructions. An attacker crafted a malicious repository whose content, when ingested by Amazon Q Developer, caused the AI agent to treat embedded instructions as legitimate commands. The agent then executed those commands within the developer's environment, including exfiltrating AWS credentials (MITRE T1552.001, Credentials In Files) to attacker-controlled infrastructure via standard application-layer communication (T1071). ReversingLabs described the scenario as analogous to a supply chain compromise (T1195.001) because the vector was repository content the developer trusted, not a direct attack on the developer's system. The command execution path maps to T1059, with the AI agent itself serving as an unwitting interpreter. The underlying weaknesses, CWE-77 (Command Injection) and CWE-74 (Injection), are well-understood in traditional application security but take on new dimensions when the injection target is an AI

agent with ambient access to developer credentials and cloud environment context. CWE-200 (Exposure of Sensitive Information) captures the downstream outcome: AWS credentials surfaced to an unauthorized party. The attack required no malware, no phishing of the developer, and no exploitation of the developer's endpoint directly. The AI agent's inherent design, reading repository content and acting on it, was the attack surface. 404media reported that the malicious payload included commands capable of wiping system data, broadening the potential impact beyond credential theft. AWS patched the vulnerability, but the disclosure reveals that no CVE identifier has been publicly assigned based on available sources, limiting structured tracking. The broader industry implication is significant: as AI coding assistants gain broader permissions and deeper integration with cloud environments, prompt injection becomes a first-class threat vector that traditional application security controls do not address by default.

Action Checklist

1. Step 1: Assess exposure, determine whether your development teams use Amazon Q Developer or any AI coding assistant with access to cloud credentials, repository content, or shell execution permissions; inventory which tools have ambient access to AWS, Azure, or GCP credential stores
2. Step 2: Review controls, audit whether developer workstations and CI/CD pipelines scope AI assistant permissions using least-privilege principles (NIST AC-6); verify that AWS IAM roles used by developers follow minimum necessary access and that credential files are not stored in plaintext locations accessible to agent processes (NIST AC-3); confirm MFA is enforced on all externally-exposed developer accounts (CIS 6.3) and remote access paths (CIS 6.4)
3. Step 3: Update threat model, add prompt injection via malicious repository content as an explicit threat scenario in your supply chain risk register; map to MITRE T1195.001 (Supply Chain Compromise: Compromise Software Dependencies) and T1552.001 (Credentials In Files); assume any AI coding agent that reads untrusted repository content is a potential injection target
4. Step 4: Communicate findings, brief engineering leadership and cloud platform owners on the specific risk that AI agents with cloud credential access can be weaponized through repository content they are asked to process; frame this as a trust-boundary problem requiring policy changes, not only a patching action
5. Step 5: Monitor developments, track AWS Security Bulletins and the ReversingLabs blog for follow-up technical indicators; watch for a CVE assignment that would enable structured tracking across vulnerability management tooling; monitor for similar disclosures affecting GitHub Copilot, Cursor, and other AI coding assistants with comparable repository-ingestion architectures

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate immediately to CISO and cloud platform owners if AWS CloudTrail or VPC Flow Logs show outbound connections from developer workstation IP ranges to non-corporate endpoints on port 443 following periods of Amazon Q Developer agent activity, or if `~/.aws/credentials` file access timestamps on developer systems postdate Amazon Q Developer process invocations with no corresponding legitimate developer action — either condition indicates likely credential exfiltration consistent with this attack pattern.

Recovery Notes	After confirming Amazon Q Developer has been updated to the patched version, rotate all AWS IAM access keys for developer accounts that ran the vulnerable version against any repository containing third-party or public code contributions, prioritizing keys with broad IAM permissions (e.g., AdministratorAccess, PowerUserAccess, or any policy granting `s3:*`, `iam:*`, or `ec2:*`). Monitor AWS CloudTrail for a minimum of 30 days post-rotation for any API calls using the rotated key IDs, which would indicate the attacker captured and attempted to use credentials before rotation. Verify that no new IAM users, access keys, or role trust-policy modifications were made from developer credentials during the exposure window by running `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=CreateUser` and equivalent queries for `CreateAccessKey` and `UpdateAssumeRolePolicy`.
Forensic Artifacts	AWS CloudTrail logs: query for `GetCallerIdentity`, `CreateAccessKey`, `AssumeRole`, `ListBuckets`, `GetObject`, and `PutObject` API calls originating from developer workstation source IPs or the IAM identity associated with the Amazon Q Developer process, filtered to the exposure window — these would reflect credential use or enumeration following exfiltration | Developer workstation filesystem: `~/.aws/credentials` and `~/.aws/config` last-access timestamps (atime) compared against Amazon Q Developer process invocation times from system logs — a read of the credentials file by a process other than the AWS CLI or SDK during an Amazon Q agent session is a direct indicator of prompt injection credential access | Developer workstation network connections: VPC Flow Logs or host-based `netstat -ano` / `ss -tunp` output capturing outbound HTTPS (port 443) connections established by the Amazon Q Developer process to non-AWS endpoints, which would represent the exfiltration channel identified in the ReversingLabs disclosure | Amazon Q Developer agent activity logs and IDE extension logs (location varies by IDE — for VS Code: `~/.vscode/extensions/amazonwebservices.amazon-q-vscode-*/` log directory): these would contain the injected repository content that triggered the malicious instruction sequence, preserving the prompt injection payload for forensic analysis | CI/CD pipeline execution logs (GitHub Actions logs, Jenkins build logs, GitLab CI job traces): for pipelines where Amazon Q Developer or CodeWhisperer agent actions were invoked, capture the full job output and environment variable dumps (excluding secret values) to identify whether malicious repository content was processed in an automated context with ambient cloud credentials bound to a service account role

Per-Action IR Details

Step 1: Assess exposure — determine whether your development teams use Amazon Q Developer or any AI coding assistant with access to cloud credentials, repository content, or shell execution permissions; inventory which tools have ambient access to AWS, Azure, or GCP credential stores

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: Establishing IR capability and asset visibility prior to an incident

Controls: CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), NIST AC-20 (Use Of External Systems)

Compensating: Run `find ~/.aws -name credentials -o -name config 2>/dev/null` on developer workstations to enumerate plaintext credential file locations; use `ps aux | grep -i 'amazon-q\|codewhisperer\|copilot\|cursor'` to identify running AI assistant processes and their effective user context; document results in a shared spreadsheet if no CMDB exists.

Evidence: This is a preparation/inventory step and does not alter live state — no volatile capture is required before execution. However, document the pre-inventory baseline: snapshot running process lists (`ps aux` or `Get-Process`) and active credential file timestamps (`ls -la ~/.aws/credentials` or `dir %USERPROFILE%\.aws\credentials`) before any remediation actions are taken, so post-incident comparison is possible.

Step 2: Review controls — audit whether developer workstations and CI/CD pipelines scope AI assistant permissions using least-privilege principles (NIST AC-6); verify that AWS IAM roles used by developers follow minimum necessary access and that credential files are not stored in plaintext locations accessible to agent processes (NIST AC-3); confirm MFA is enforced on all externally-exposed developer accounts (CIS 6.3) and remote access paths (CIS 6.4)

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: Preventing incidents through secure configuration and access control hardening

Controls: NIST AC-3 (Access Enforcement), NIST AC-6 (Least Privilege), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.4 (Require MFA for Remote Network Access), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts)

Compensating: Use the AWS CLI to audit IAM inline and managed policies attached to developer roles: `aws iam list-attached-role-policies --role-name ` and `aws iam get-policy-version` to enumerate permissions; run `aws iam generate-credential-report && aws iam get-credential-report` to identify accounts without MFA; search CI/CD configuration files (`.github/workflows/*.yml`, `Jenkinsfile`, `.gitlab-ci.yml`) with `grep -r 'AWS_ACCESS_KEY_ID\|AWS_SECRET_ACCESS_KEY' .` to find hardcoded or ambient credentials accessible to pipeline steps that invoke Amazon Q.

Evidence: This step reviews configuration and does not revoke credentials or alter live state — no volatile pre-capture is mandatory. Before modifying any IAM policy or MFA enforcement setting, export current IAM state: `aws iam get-account-authorization-details > iam_baseline_$(date +%Y%m%d).json` to preserve a point-in-time snapshot for forensic comparison if credential misuse is later discovered.

Step 3: Update threat model — add prompt injection via malicious repository content as an explicit threat scenario in your supply chain risk register; map to MITRE T1195.001 (Supply Chain Compromise: Compromise Software Dependencies) and T1552.001 (Credentials In Files); assume any AI coding agent that reads untrusted repository content is a potential injection target

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Lessons learned, threat model updates, and detection improvement based on incident findings

Controls: NIST AC-20 (Use Of External Systems), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Document the prompt injection threat scenario as a one-page entry in the risk register using free templates from OWASP Top 10 for LLM Applications (specifically LLM01: Prompt Injection); create a standing agenda item in monthly security reviews to track newly disclosed AI coding assistant vulnerabilities affecting Amazon Q Developer, GitHub Copilot, Cursor, and comparable tools that ingest untrusted repository content.

Evidence: This is a documentation and planning step that does not alter live system state — no volatile evidence capture is required. Reference the ReversingLabs research disclosure and AWS Security Bulletin (when published) as the evidentiary basis for the threat scenario entry; retain these source documents in the risk register record for audit traceability.

Step 4: Communicate findings — brief engineering leadership and cloud platform owners on the specific risk that AI agents with cloud credential access can be weaponized through repository content they are asked to process; frame this as a trust-boundary problem requiring policy changes, not only a patching action

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Sharing lessons learned with stakeholders and driving policy improvements across the organization

Controls: NIST AC-1 (Policy And Procedures)

Compensating: Prepare a one-page technical brief using the ReversingLabs disclosure narrative as the concrete example: an AI agent (Amazon Q Developer) ingested attacker-controlled repository content containing embedded

instructions that redirected the agent to exfiltrate `~/.aws/credentials` to an external endpoint; distribute via email with a read-receipt request to establish documented notification for compliance purposes.

Evidence: This is a communication step and does not alter live system state — no volatile evidence capture is required before execution. Retain distribution records (email receipts, meeting minutes) as documentation that responsible parties were notified, which supports regulatory and audit requirements if a related credential exposure is later discovered.

Step 5: Monitor developments — track AWS Security Bulletins and the ReversingLabs blog for follow-up technical indicators; watch for a CVE assignment that would enable structured tracking across vulnerability management tooling; monitor for similar disclosures affecting GitHub Copilot, Cursor, and other AI coding assistants with comparable repository-ingestion architectures

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Improving detection capabilities and integrating threat intelligence to prevent recurrence

Controls: CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), NIST AU-6 (Audit Record Review, Analysis, And Reporting)

Compensating: Subscribe to the AWS Security Bulletins RSS feed (https://aws.amazon.com/security/security-bulletins/ — validate this URL before use) and configure a free RSS-to-email bridge (e.g., Blogtrottr) to alert on new Amazon Q Developer advisories; set up a Google Alert for 'Amazon Q Developer vulnerability' and 'AI coding assistant prompt injection' to capture community disclosures; create a YARA rule or Sigma rule searching outbound HTTP POST traffic from developer workstations to non-corporate endpoints originating from processes matching Amazon Q Developer's executable name, as a behavioral indicator of credential exfiltration consistent with this attack pattern.

Evidence: This is an ongoing monitoring and intelligence step and does not alter live system state — no volatile evidence capture is required. If a CVE is assigned, immediately cross-reference it against AWS CloudTrail logs querying for `GetCallerIdentity`, `AssumeRole`, or `ListBuckets` API calls from developer workstation IP ranges at anomalous hours, as these would indicate AWS credential use following a successful prompt injection exfiltration from Amazon Q Developer.

Detection Guidance

Hunt for anomalous outbound network connections originating from developer workstations or CI/CD runners during periods when AI coding assistants are active, particularly to domains not associated with AWS service endpoints (AU-6, AU-12, Audit Record Review and Generation). Review AWS CloudTrail logs for credential usage events, GetSessionToken, AssumeRole, or API calls, originating from unexpected source IPs or geographic locations that do not match developer baselines; this addresses T1552.001 exfiltration of credentials and subsequent use (NIST AU-6). Enable and review AWS IAM Access Analyzer alerts for credentials used outside expected network perimeters. Monitor for unusual DNS queries or HTTP POST requests from developer machines to unfamiliar external destinations during active coding sessions (T1071, Application Layer Protocol). Audit AI assistant plugin and extension permissions across your developer toolchain; flag any tool with read access to credential files, shell execution rights, or network egress that lacks approval in your software inventory (CIS 2.1, Establish and Maintain a Software Inventory). Review repository content ingested by AI agents for embedded instruction patterns targeting agent behavior, such as markdown or code comments containing imperative commands directed at an AI system rather than human readers. Apply D3-LAM (Local Account Monitoring) to detect credential use anomalies on developer accounts. Apply D3-UAP (User Account Permissions) review to confirm AI agent processes run under appropriately scoped accounts, not developer-level or admin-level identities. Apply D3-CRO (Credential Rotation) immediately for any developer who used Amazon Q Developer against external or unvetted repositories before the patch was applied.

Indicators of Compromise

Type	Value	Context	Confidence
TOOL	`Amazon Q Developer AI agent`	AI coding agent leveraged via malicious repository content injection to execute attacker-supplied commands and exfiltrate AWS developer credentials to attacker-controlled infrastructure	HIGH
URL	`Pending — refer to ReversingLabs blog post 'How AWS averted an AI coding supply chain disaster' (https://www.reversinglabs.com/blog/aws-amazonq-ai-incident) for any published indicators`	ReversingLabs published a detailed incident analysis; specific C2 domains, payload hashes, or network indicators, if any were released, appear in that report	LOW

Framework Mappings

MITRE-ATTACK

- **T1552.001** — Credentials In Files
- **T1071** — Application Layer Protocol
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-7** — Software, Firmware, and Information Integrity
- **AC-3** — Access Enforcement
- **SC-28** — Protection of Information at Rest
- **SI-10** — Information Input Validation
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A03:2021** — Injection

HIPAA-SECURITY

- **164.312(a)(1)** — Access Control

- **164.312(d)** — Person or Entity Authentication

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **6.3** — Require MFA for Externally-Exposed Applications
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management
- **15.1** — Establish and Maintain an Inventory of Service Providers

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1552.001	Credentials In Files	Credential-Access
T1071	Application Layer Protocol	Command-And-Control
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution

Sources

Source	URL	Tier
Amazon Q: Now with Helpful AI-Powered Self-Destruct Capabilities	https://www.lastweekinaws.com/blog/amazon-q-now-with-helpful-ai-pow...	T3
Coding Assistant - Amazon Q Developer	https://aws.amazon.com/q/developer/	T3

Source	URL	Tier
How AWS averted an AI coding supply chain disaster | RL Blog	https://www.reversinglabs.com/blog/aws-amazonq-ai-incident	T3
Amazon's AI coding assistant exposed nearly 1 million ... - Reddit	https://www.reddit.com/r/webdev/comments/1maynl7/amazons_ai_coding_...	T3
Hacker Plants Computer 'Wiping' Commands in Amazon's AI Coding ...	https://www.404media.co/hacker-plants-computer-wiping-commands-in-a...	T3

Generated 2026-06-29 07:11 UTC by TJS Security Command Center