INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-15 19:00 UTC

AI Writes the Code, AI Finds the Flaws: A Closing Attack Window Security Teams Cannot Ignore

SECURITY ANALYSIS | HIGH | CVSS 5.0

SCC Item ID	SCC-STY-2026-0132
Type	Security Analysis
Severity	HIGH
CVSS Base Score	5.0
Affected Products	Broad, AI coding assistants (e.g., GitHub Copilot, Cursor, Codeium), AI agent frameworks (e.g., LangChain, AutoGPT, CrewAI); no single named product
Published	2026-05-18T09:00:00
Discovery Source	Rss

Executive Summary

AI coding assistants are accelerating software development at scale, but they are simultaneously introducing exploitable flaws, including SQL injection, OS command injection, and missing authentication, faster than human review processes can catch them. Autonomous AI agents have now demonstrated the ability to discover and act on these vulnerabilities without human direction, significantly compressing the window between deployment and exploitation. This structural asymmetry between machine-speed offense and human-speed defense represents a durable shift in the threat landscape, one that demands changes to how organizations govern AI-assisted development, not just how they patch.

Technical Analysis

The threat dynamic documented in this story is not a single incident but a structural condition emerging from two converging trends: the mass adoption of AI coding assistants and the rise of autonomous AI agents capable of offensive action.

On the generation side, tools like GitHub Copilot, Cursor, and Codeium produce code at volumes no manual review process was designed to absorb. Veracode research indicates that AI-generated code exhibits improper input validation (CWE-20), OS command injection (CWE-78), SQL injection (CWE-89), missing authentication (CWE-306), and insecure deserialization (CWE-502). These are not novel attack surfaces; they are foundational weakness categories that appear in the OWASP Top 10 and MITRE's CWE corpus year after year. What AI generation changes is throughput: the same flaw that a developer might introduce once a week now enters codebases at the pace of AI suggestion acceptance.

On the exploitation side, Microsoft's Security Blog (May 2026) documents a qualitatively different problem: AI agent frameworks including LangChain, AutoGPT, and CrewAI carry remote code execution vulnerabilities where attacker-controlled prompts become shell execution vectors. The Microsoft research maps directly to MITRE ATT&CK T1059 (Command and Scripting Interpreter) and T1190 (Exploit Public-Facing Application), with the novel element being that the execution path runs through the AI agent's own orchestration layer rather than a conventional application binary. An arXiv preprint from May 2026 frames this architecturally: AI agents interact with operating system resources, execute code, and manage credentials in ways that resemble OS-level privilege, but without the decades of security hardening applied to actual operating systems.

The MITRE techniques present across this threat surface, T1595 (Active Scanning), T1590 (Gather Victim Network Information), T1651 (Cloud Administration Command), and T1195.001 (Compromise Software Dependencies), suggest that autonomous agents are not just exploiting individual flaws but are capable of conducting multi-stage reconnaissance and supply chain interference at machine speed.

The defensive gap this creates is a velocity problem. Static analysis tools, peer review gates, and penetration testing cycles operate on human timescales. When code generation and vulnerability discovery both move to machine speed, the interval between a flaw entering production and an agent identifying it for exploitation shrinks toward the point where traditional controls provide little practical protection. Detection engineering and code review processes designed for quarterly release cadences are structurally mismatched against CI/CD pipelines fed by AI assistants.

Action Checklist

1. Step 1: Assess exposure, audit which AI coding assistants (GitHub Copilot, Cursor, Codeium, or equivalents) are authorized or in shadow use across your development teams; inventory any AI agent frameworks (LangChain, AutoGPT, CrewAI) deployed in production or test environments
2. Step 2: Review controls, verify that SAST and SCA tooling is integrated at the PR/merge gate level, not as a post-deployment scan; confirm that AI-generated code is not bypassing review queues due to velocity pressure; validate that AI agent frameworks are sandboxed with least-privilege OS and network access
3. Step 3: Update threat model, add AI-assisted vulnerability discovery as an adversary capability to your threat register; map T1059, T1190, and T1195.001 to your current detection coverage and identify gaps specific to AI agent execution paths
4. Step 4: Communicate findings, brief engineering leadership and the CISO on the throughput mismatch between AI code generation rates and current review capacity; frame this as a process governance issue, not solely a tooling issue
5. Step 5: Monitor developments, track Microsoft Security Blog and CISA advisories for follow-up disclosures on AI agent framework vulnerabilities; monitor NVD and security vendor disclosures for CWE-20, CWE-78, CWE-89, CWE-306, and CWE-502 findings in AI tooling packages and dependencies

IR / Forensic Enrichment

Triage Priority	URGENT

Escalation Criteria	Escalate immediately if SAST gate review confirms that AI-generated code containing CWE-89, CWE-78, or CWE-306 findings has already been merged to production and is reachable by unauthenticated external requests, or if osquery or Sysmon telemetry shows an AI agent framework process (LangChain, AutoGPT, CrewAI) spawning unexpected child processes or making outbound connections to non-approved endpoints, as either condition indicates active exploitation risk requiring IR plan activation per NIST IR-4 (Incident Handling) and potential breach notification assessment if PII/PHI is accessible via the vulnerable endpoints.
Recovery Notes	Following any containment action (e.g., disabling a vulnerable AI-generated endpoint or isolating an AI agent framework container), verify recovery by re-running Semgrep OSS with the OWASP Top Ten ruleset against the full codebase branch and confirming zero findings for CWE-89, CWE-78, CWE-306, and CWE-502 before re-enabling the endpoint or restoring agent access. Monitor WAF logs (ModSecurity audit log) and web server access logs for a minimum of 30 days post-recovery for renewed probe patterns targeting the previously vulnerable URI paths, as autonomous AI-assisted scanners may re-discover repaired endpoints if the patch is incomplete. Update the threat register and IR plan with the confirmed attack path within 5 business days per NIST 800-61r3 §4 post-incident activity requirements.
Forensic Artifacts	Web server access logs (Apache /var/log/apache2/access.log or Nginx /var/log/nginx/access.log) filtered for URI patterns containing SQL metacharacters (' OR 1=1, UNION SELECT, -- -) or OS command delimiters (; | && `cmd`) in query strings or POST bodies — these are the primary forensic record of CWE-89/CWE-78 exploitation attempts against AI-generated vulnerable endpoints | CI/CD pipeline execution logs (GitHub Actions workflow run logs, GitLab job logs, Jenkins build console output) for the specific build jobs that processed AI-generated code PRs — these establish which SAST gates ran, which were bypassed, and which AI-introduced flaws were detected or missed before merge to production | AI agent framework process execution records: Sysmon Event ID 1 (Process Create) logs on Windows or Linux auditd 'execve' syscall logs showing the LangChain, AutoGPT, or CrewAI runtime process tree, including any child processes (bash, sh, python, curl, wget) spawned during autonomous task execution that may indicate autonomous vulnerability scanning or exploit attempt behavior (MITRE T1059) | Python package installation audit trail: pip install logs (/root/.local/share/pip/log/ or equivalent) and virtual environment site-packages directories timestamped to capture when AI framework packages (langchain, autogpt, crewai and their transitive dependencies) were installed, which versions were present at time of any incident, and whether any deserialization-vulnerable packages (CWE-502) such as pickle-dependent libraries were included | GitHub or GitLab audit log exports filtered for 'push_protection_bypass', 'required_status_check_override', and 'merge' events on repositories where AI coding assistants were active — these records establish whether AI-generated code bypassed mandatory SAST review gates under velocity pressure, which is the primary governance failure mechanism in this threat scenario (MITRE T1195.001)

Per-Action IR Details

Step 1: Assess exposure — audit which AI coding assistants (GitHub Copilot, Cursor, Codeium, or equivalents) are authorized or in shadow use across your development teams; inventory any AI agent frameworks (LangChain, AutoGPT, CrewAI) deployed in production or test environments

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: Establishing IR capability includes maintaining an accurate inventory of tools and services that introduce risk surface, including AI-assisted development tooling and autonomous agent frameworks

Controls: NIST IR-4 (Incident Handling) — preparation sub-phase requires knowing what assets and tooling exist before an incident occurs, NIST SI-5 (Security Alerts, Advisories, and Directives) — receiving and acting on advisories about AI tooling risks requires knowing which tools are deployed, CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory) — inventory must include AI coding assistants and agent runtimes as software assets with network and code-access exposure, CIS 2.1 (Establish and Maintain a Software Inventory) — AI coding assistant plugins (Copilot IDE extension, Codeium, Cursor) and agent framework packages (LangChain, AutoGPT, CrewAI) must appear in the authorized software inventory, CIS 2.3 (Address Unauthorized Software) — shadow use of unapproved AI coding assistants constitutes unauthorized software and must be discovered and remediated on a weekly basis

Compensating: Run 'pip list' or 'pip freeze' across developer workstations and CI/CD build agents via a scheduled osquery query (SELECT name, version FROM python_packages WHERE name LIKE '%langchain%' OR name LIKE '%autogpt%' OR name LIKE '%crewai%' OR name LIKE '%openai%') to surface AI framework installations. For IDE extension inventory, query VSCode extension directories: on Linux/macOS 'ls ~/.vscode/extensions | grep -iE "copilot|codeium|cursor"'; on Windows 'dir %USERPROFILE%\.vscode\extensions | findstr /i "copilot codeium"'. Cross-reference npm global installs with 'npm list -g --depth=0' for JavaScript-based agent tooling. A 2-person team can script and schedule these as weekly cron jobs feeding a flat CSV inventory.

Evidence: Before beginning the audit, snapshot the current state to establish a baseline: export the full list of VSCode/JetBrains extensions from developer endpoints; capture 'pip freeze' and 'npm list -g' output from build agents and developer machines; pull GitHub OAuth app authorization logs (Settings > Developer Settings > Authorized OAuth Apps) to identify Copilot and third-party AI tool authorizations; retrieve cloud IAM role bindings for any service accounts associated with LangChain or AutoGPT deployments to document existing privilege grants before any remediation changes the state.

Step 2: Review controls — verify that SAST and SCA tooling is integrated at the PR/merge gate level, not as a post-deployment scan; confirm that AI-generated code is not bypassing review queues due to velocity pressure; validate that AI agent frameworks are sandboxed with least-privilege OS and network access

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: Prevention controls implemented prior to incidents reduce the frequency and severity of incidents; verifying SAST gate enforcement and agent sandboxing directly reduces the AI-generated flaw introduction rate

Controls: NIST SI-2 (Flaw Remediation) — SAST integrated at PR gate is the primary automated mechanism for identifying AI-introduced flaws (CWE-89 SQL injection, CWE-78 OS command injection, CWE-306 missing authentication) before they reach production, NIST SI-4 (System Monitoring) — AI agent framework runtimes (LangChain, AutoGPT, CrewAI) must be monitored for anomalous OS and network calls consistent with autonomous vulnerability discovery behavior, NIST SI-7 (Software, Firmware, and Information Integrity) — integrity verification of the CI/CD pipeline configuration ensures SAST gates have not been disabled or bypassed under velocity pressure, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — SAST/SCA at the PR gate is the operationalized form of vulnerability management for AI-generated code at machine generation speed, CIS 4.4 (Implement and Manage a Firewall on Servers) — AI agent framework containers and processes must be network-isolated with explicit egress allow-lists to prevent autonomous agents from reaching external exploit databases or C2 infrastructure

Compensating: For SAST without enterprise tooling, configure Semgrep OSS (free) as a GitHub Actions or GitLab CI step using the 'p/owasp-top-ten' and 'p/python' rulesets, which cover CWE-89, CWE-78, and CWE-306 patterns commonly introduced by AI code generation. Add a required status check in GitHub branch protection rules so Semgrep must pass before merge. For SCA, integrate OWASP Dependency-Check as a pipeline step targeting LangChain, CrewAI, and AutoGPT dependency trees. For agent sandboxing, use Linux network namespaces or Docker's '--network none' flag combined with explicit '--cap-drop ALL' to restrict AI agent containers; verify with 'docker inspect | grep -A5 NetworkMode' and 'docker inspect | grep CapAdd'.

Evidence: Capture the current CI/CD pipeline configuration files (e.g., .github/workflows/*.yml, .gitlab-ci.yml, Jenkinsfile) before making any changes — these establish whether SAST gates existed and whether branch protection rules required them. Pull GitHub audit logs (Organization > Audit Log, filter by 'protected_branch' events) to identify any recent disabling of required status checks. For agent sandboxing, capture 'docker inspect' output and 'iptables -L -n' or 'nft list ruleset' from agent host systems to document current network egress posture. These records establish the

pre-remediation state needed for post-incident timeline reconstruction if an AI-generated flaw is exploited.

Step 3: Update threat model — add AI-assisted vulnerability discovery as an adversary capability to your threat register; map T1059, T1190, and T1195.001 to your current detection coverage and identify gaps specific to AI agent execution paths

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: Understanding adversary TTPs and mapping them to detection coverage is prerequisite to identifying adverse events; AI agent-assisted exploitation compresses detection windows requiring proactive gap analysis against T1059, T1190, and T1195.001

Controls: NIST IR-4 (Incident Handling) — threat register updates and TTP coverage mapping are preparation activities that directly enable detection and analysis of AI-assisted exploitation attempts, NIST RA-3 (Risk Assessment) — adding AI-assisted autonomous vulnerability discovery as an adversary capability constitutes a material change to the threat landscape requiring formal risk register update, NIST SI-4 (System Monitoring) — detection coverage gaps identified for T1059 (Command and Scripting Interpreter used by AI agents for autonomous code execution), T1190 (Exploit Public-Facing Application targeting AI-generated vulnerable endpoints), and T1195.001 (Compromise Software Dependencies — AI framework supply chain) must be closed with specific monitoring controls, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — threat modeling update operationalizes vulnerability management by prioritizing AI-generated CWE-89/78/306/502 findings based on AI-assisted adversary discovery speed, CIS 8.2 (Collect Audit Logs) — closing detection gaps for T1059 and T1190 requires confirming audit logs are collected from web application servers, API gateways, and agent runtime environments where AI-introduced flaws would be exploited

Compensating: Map existing log sources against the three ATT&CK techniques using the free MITRE ATT&CK Navigator (https://mitre-attack.github.io/attack-navigator/) — export the coverage layer as JSON to document the gap baseline. For T1190 detection against AI-generated SQLi/CMDi endpoints, deploy the free ModSecurity WAF with the OWASP CRS ruleset and enable paranoia level 2; review '/var/log/modsec_audit.log' for rule IDs 942100 (SQL injection) and 932100-932180 (OS command injection). For T1059 detection of AI agent script execution, deploy Sysmon with a configuration targeting process creation events (Event ID 1) where parent processes match known AI agent runtime names (python.exe, node, uvicorn) spawning shell interpreters (cmd.exe, bash, sh, powershell.exe). Use the free Sigma rule 'proc_creation_win_susp_script_execution' as a starting template.

Evidence: Before updating the threat model, gather current detection baseline evidence: export existing SIEM or log aggregation alert rules covering T1059, T1190, and T1195 to document pre-existing coverage; pull the last 30 days of WAF/IDS alert logs filtered for SQLi (CWE-89) and command injection (CWE-78) patterns to identify whether AI-generated vulnerable endpoints have already been probed; retrieve the dependency manifest files (requirements.txt, package.json, go.mod) for any production-deployed AI agent frameworks to assess T1195.001 supply chain exposure. This evidence establishes the detection gap baseline and supports scope estimation per NIST 800-61r3 §3.2 adverse event analysis guidance.

Step 4: Communicate findings — brief engineering leadership and the CISO on the throughput mismatch between AI code generation rates and current review capacity; frame this as a process governance issue, not solely a tooling issue

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: IR capability requires organizational commitment and resource allocation; communicating the structural asymmetry between AI-speed code generation and human-speed review capacity is a governance action that precedes effective incident prevention and response

Controls: NIST IR-1 (Policy and Procedures) — the throughput mismatch between AI code generation and review capacity is a policy gap requiring executive acknowledgment and documented policy update governing AI coding assistant use in the SDLC, NIST IR-8 (Incident Response Plan) — the IR plan must be updated to address AI-assisted exploitation scenarios; briefing the CISO initiates the authorization process for that update, NIST IR-6 (Incident Reporting) — establishing a reporting channel for AI-generated code quality findings to leadership formalizes the organizational feedback loop needed to resource the process governance fix, CIS 7.2 (Establish and Maintain a Remediation Process) — the CISO brief must include a risk-based remediation strategy that accounts for AI code generation velocity as a factor determining remediation urgency thresholds

Compensating: For a resource-constrained team, build a one-page brief using concrete metrics rather than qualitative risk language: calculate the ratio of AI-assisted PRs merged per week (pull from GitHub Insights or GitLab analytics) against average SAST finding resolution time (pull from Semgrep or Dependency-Check historical output). Present the delta as mean time to review versus mean time for an autonomous agent to scan and identify a newly deployed endpoint — reference the published academic findings (e.g., Fang et al. 2024) showing AI agents exploiting one-day vulnerabilities at high success rates to ground the urgency. This framing converts an abstract risk into a measurable throughput gap that non-security executives can evaluate for resource allocation decisions.

Evidence: Assemble supporting evidence for the brief before the meeting: pull GitHub or GitLab merge statistics showing volume of AI-assisted commits (Copilot contribution data is available via GitHub Copilot usage API for organizations); collect any existing SAST scan results showing CWE-89, CWE-78, CWE-306, or CWE-502 findings in recently merged AI-generated code to demonstrate the flaw introduction rate is not hypothetical; document any instances where PR velocity pressure resulted in bypassed or waived SAST gates (GitHub audit log, filter 'bypass_push_protection' or 'required_status_checks' events). This evidence transforms the communication from a predictive risk argument into a data-backed governance finding.

Step 5: Monitor developments — track Microsoft Security Blog and CISA advisories for follow-up disclosures on AI agent framework vulnerabilities; subscribe to NVD feeds for CWE-20, CWE-78, CWE-89, CWE-306, and CWE-502 findings in AI tooling packages

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Lessons learned and continuous improvement include updating detection capabilities and threat intelligence subscriptions based on evolving adversary capabilities; monitoring NVD CWE feeds and CISA advisories for AI agent framework vulnerabilities operationalizes this for the AI-assisted exploitation threat class

Controls: NIST SI-5 (Security Alerts, Advisories, and Directives) — formal subscription to NVD CWE-filtered feeds and CISA KEV for AI framework packages (LangChain, AutoGPT, CrewAI) is the operationalized implementation of this control for this threat class, NIST IR-4 (Incident Handling) — continuous monitoring of emerging AI agent vulnerability disclosures feeds back into the preparation phase, ensuring the handling capability evolves at the same pace as AI-assisted adversary tooling, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — threat intelligence from NVD and CISA must be correlated with internal audit records to identify whether disclosed AI framework CVEs affect packages already deployed in production, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — NVD CWE feed subscriptions for CWE-20 (Improper Input Validation), CWE-78 (OS Command Injection), CWE-89 (SQL Injection), CWE-306 (Missing Authentication), and CWE-502 (Deserialization) in AI tooling packages are the continuous intelligence input to the vulnerability management process, CIS 7.3 (Perform Automated Operating System Patch Management) — new CVEs in AI agent framework OS dependencies disclosed via monitored feeds must trigger automated patch evaluation within the documented SLA

Compensating: Configure a free NVD API feed subscription filtered by CWE using the NVD REST API endpoint 'https://services.nvd.nist.gov/rest/json/cves/2.0?cweId=CWE-89' (repeat for CWE-78, CWE-20, CWE-306, CWE-502) and pipe results through a simple Python script using the 'nvdlib' package to filter on keywords 'langchain', 'autogpt', 'crewai', 'copilot', 'codeium', 'openai' in the CPE or description fields — schedule as a daily cron job with email alert output. Subscribe to the CISA KEV RSS feed (https://www.cisa.gov/sites/default/files/feeds/known_exploited_vulnerabilities.json) using a free RSS-to-email service or a scheduled curl/jq pipeline. Follow the Microsoft Security Response Center (MSRC) blog RSS for Copilot and GitHub-related disclosures. A 2-person team can operationalize this monitoring stack in under a day with no licensing cost.

Evidence: Establish the monitoring baseline before new disclosures arrive: export the current production dependency manifests (requirements.txt, package-lock.json, Gemfile.lock) for all AI agent framework deployments and run them through 'pip-audit' or OWASP Dependency-Check to capture the zero-day baseline of known vulnerabilities already present; archive these results with timestamps so that when a new CVE is disclosed for a monitored CWE, you can immediately determine whether the affected version is deployed without re-scanning from scratch. Retain NVD API query results and CISA KEV snapshots on a weekly basis to support post-incident timeline reconstruction if an AI framework vulnerability is later confirmed exploited.

Detection Guidance

Detection for this threat surface spans three layers: code pipeline, runtime behavior, and agent orchestration.

In the code pipeline: Configure SAST tooling to flag CWE-20, CWE-78, CWE-89, CWE-306, and CWE-502 findings specifically in files modified by AI assistant commits (many platforms tag these). Track the ratio of AI-assisted lines merged to findings flagged, a rising merge rate with a flat or falling finding rate may indicate review bypass rather than genuine quality improvement.

At runtime: Hunt for unexpected shell spawning from processes associated with AI agent frameworks (Python interpreters running LangChain or AutoGPT components, Node.js processes running agent orchestration layers). Alert on child process creation from these parent processes, particularly where the child is cmd.exe, sh, bash, or a scripting interpreter. Log and alert on outbound connections from AI agent processes to unexpected external endpoints, consistent with T1590 and T1595 reconnaissance behavior.

In the agent orchestration layer: Review prompt logging if your AI agent deployment supports it. Look for prompt inputs that include shell metacharacters, SQL syntax fragments, or instruction-injection patterns (e.g., 'ignore previous instructions'). Microsoft's May 2026 research identifies prompt-to-shell execution as the primary RCE vector, any agent that can invoke OS commands should have that capability explicitly scoped and monitored.

Policy audit: Verify that AI agent service accounts operate under least-privilege principles and are not granted administrative cloud roles. T1651 (Cloud Administration Command) is a realistic escalation path if an agent with broad IAM permissions is compromised via prompt injection.

Framework Mappings

MITRE-ATTACK

- **T1190** — Exploit Public-Facing Application
- **T1203** — Exploitation for Client Execution
- **T1590** — Gather Victim Network Information
- **T1595** — Active Scanning
- **T1651** — Cloud Administration Command
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter

NIST-800-53R5

- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CA-7** — Continuous Monitoring
- **CM-7** — Least Functionality

- **SI-10** — Information Input Validation
- **IA-2** — Identification and Authentication (Organizational Users)
- **AT-2** — Literacy Training and Awareness

OWASP-TOP10-2021

- **A03:2021** — Injection
- **A08:2021** — Software and Data Integrity Failures
- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **2.5** — Allowlist Authorized Software
- **6.3** — Require MFA for Externally-Exposed Applications
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **8.2** — Collect Audit Logs

ISO-27001-2022

- **A.8.26** — Application security requirements
- **A.8.28** — Secure coding
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1190	Exploit Public-Facing Application	Initial-Access
T1203	Exploitation for Client Execution	Execution
T1590	Gather Victim Network Information	Reconnaissance
T1595	Active Scanning	Reconnaissance
T1651	Cloud Administration Command	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution

Sources

Source	URL	Tier
Security News	https://www.darkreading.com/cyber-risk/ai-code-and-agents-forces-de...	T3
When prompts become shells: RCE vulnerabilities in AI agent ...	https://www.microsoft.com/en-us/security/blog/2026/05/07/prompts-be...	T1
Toward Securing AI Agents Like Operating Systems - arXiv	https://arxiv.org/html/2605.14932v1	T2
Why AI Coding Tools Are Creating Security Gaps - Veracode	https://www.veracode.com/blog/ai-coding-tools-security-gaps/	T3
AI tool poisoning exposes a major flaw in enterprise agent security	https://www.linkedin.com/posts/allenmark_ai-tool-poisoning-exposes-...	T3

Generated 2026-05-15 19:00 UTC by TJS Security Command Center