INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-14 04:25 UTC

Agentjacking: Novel Attack Class Hijacks AI Coding Agents via Log Poisoning and Prompt Injection

SECURITY ANALYSIS | HIGH | CVSS 8.1

SCC Item ID	SCC-STY-2026-0196
Type	Security Analysis
Severity	HIGH
CVSS Base Score	8.1
Affected Products	OpenClaw AI agent framework; integrations with Sentry error-tracking platform; broadly applicable to AI coding agents with tool-use and log-ingestion capabilities
Published	2026-06-12
Discovery Source	Gemini

Executive Summary

A newly documented attack class called 'Agentjacking' allows attackers to hijack AI coding agents by embedding malicious instructions inside application logs, error-tracking feeds, and dependency outputs that agents ingest as part of normal operation. Because these agents routinely hold access to codebases, secrets, CI/CD pipelines, and external APIs, a single successful injection can give an attacker broad lateral movement across a developer environment without triggering conventional perimeter controls. This research signals a structural security gap in how organizations govern AI agent privilege and trust boundaries, a gap that will widen as agentic AI adoption accelerates across software development workflows.

Technical Analysis

Agentjacking represents a category of indirect prompt injection attacks specifically targeting AI coding agents with tool-use and log-ingestion capabilities. The core mechanism, documented by Eye Security and formalized in a systematic taxonomy published on arXiv (2603.27517), exploits the fundamental design assumption that content arriving through trusted internal channels, application logs, error-tracking platforms such as Sentry, dependency resolution outputs, is safe to process as operational context.

The attack chain begins with an adversary planting a crafted payload inside a data source the AI agent is expected to read. When the agent ingests that content during a routine task, reviewing a stack trace, parsing a build log, querying an error-tracking platform, it interprets the embedded instructions as legitimate directives and

executes them. Demonstrated outcomes in the OpenClaw research include execution of attacker-controlled code, exfiltration of secrets and environment variables, and pivoting within developer infrastructure.

The OpenClaw AI agent framework served as the primary research subject. Community disclosure on Reddit corroborated findings independently, suggesting the attack surface is not unique to one implementation but reflects broader architectural patterns across agentic frameworks.

Three defensive gaps compound the risk. First, AI agents typically operate with elevated privilege by design, they need broad access to be useful, which means the blast radius of a successful injection is large. Second, traditional input validation is bypassed entirely because the poisoned content does not arrive via direct user input; it arrives through channels the agent has been configured to trust. Third, most organizations lack audit logging or behavioral monitoring specific to AI agent actions, meaning agent-driven lateral movement may not generate the alerts that human-initiated actions would.

MITRE ATT&CK techniques mapped to this attack class include T1190 (Exploit Public-Facing Application), T1059 (Command and Scripting Interpreter), T1574 (Hijack Execution Flow), and T1552 (Unsecured Credentials). CWEs involved are CWE-77 (Command Injection), CWE-20 (Improper Input Validation), CWE-94 (Code Injection), and CWE-116 (Improper Encoding or Escaping of Output). Sources for this analysis are rated secondary tier; primary authoritative framework documentation does not yet address AI agent-specific attack patterns, reflecting how recently this threat class has emerged.

Action Checklist

1. Step 1: Assess AI agent deployment, inventory every AI coding agent operating in your environment, including OpenClaw or similar agentic frameworks, and document what data sources each agent is permitted to ingest (logs, error-tracking platforms, dependency feeds, external APIs).
2. Step 2: Apply least privilege to agent accounts, enforce NIST AC-6 (Least Privilege) and CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts) by scoping AI agent permissions to the minimum required for each task; remove standing access to secrets stores, CI/CD pipelines, and production systems where not operationally necessary.
3. Step 3: Enforce MFA and access controls on developer tooling, require MFA on all systems AI agents can reach or authenticate against, per CIS 6.3 (Require MFA for Externally-Exposed Applications) and CIS 6.5 (Require MFA for Administrative Access); apply defense-in-depth multifactor authentication and user account permission controls to agent service accounts.
4. Step 4: Audit log ingestion pipelines, review which external or third-party data sources (e.g., Sentry, dependency registries) feed into AI agent context windows; validate that content from these sources is treated as untrusted input, not as authoritative instructions, per NIST AC-4 (Information Flow Enforcement).
5. Step 5: Enable audit logging for AI agent actions, implement NIST AU-2 (Event Logging) and AU-12 (Audit Record Generation) to capture agent-initiated file access, code execution, secret retrieval, and API calls as discrete auditable events; configure AU-6 (Audit Record Review, Analysis, and Reporting) to alert on anomalous agent behavior.
6. Step 6: Update threat model, add indirect prompt injection via log poisoning as a named attack pattern in your threat register, mapped to T1059 and T1552, and assess whether existing detection rules cover agent-driven command execution and credential access.

7. Step 7: Monitor for follow-up research and vendor response, track updates to the arXiv taxonomy (2603.27517), Eye Security blog, and reco.ai disclosures; watch for patches or architectural guidance from AI agent framework maintainers including OpenClaw.

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate immediately to senior IR leadership and legal/compliance if audit evidence from Steps 4 or 5 indicates an AI agent has already executed instructions sourced from a poisoned log entry — specifically, any agent-initiated access to secrets stores, CI/CD credential files, or outbound API calls to non-whitelisted endpoints — as this constitutes confirmed lateral movement that may trigger breach notification obligations if developer credentials, source code, or production secrets were exposed.
Recovery Notes	Before restoring normal agent operations, re-provision all agent service accounts with freshly scoped tokens (do not rotate-in-place tokens that were active during the suspected compromise window) and re-validate that OpenClaw or equivalent framework configuration enforces a clear trust boundary between ingested log content and the agent's instruction context. Monitor agent process trees and outbound API calls continuously for at least 14 days post-recovery using the auditd or Sysmon rules deployed in Step 5, paying particular attention to any recurrence of child shell spawning or access to .aws/credentials, .ssh/, or CI/CD token files. Treat any reinstated Sentry or dependency feed integration as untrusted until the vendor confirms architectural mitigations against indirect prompt injection are in place.
Forensic Artifacts	OpenClaw session history files (commonly ~/.openclaw/sessions/*.json or equivalent): these persist the agent's context window turn-by-turn, including tool call inputs and outputs — a successful agentjacking event will appear as an instruction in the 'assistant' or 'tool_result' role sourced from a log ingestion tool call, followed immediately by anomalous file-access or API-call tool invocations the human operator did not initiate. | Sentry raw event export (JSON) for the 7 days preceding detection: specifically the 'message', 'title', 'breadcrumbs.values[].message', and 'contexts' fields of error events ingested by the agent — prompt injection payloads embedded by attackers in exception messages or breadcrumb entries will appear verbatim in these fields alongside the legitimate stack trace data. | Agent process tree and child process audit records: auditd EXECVE syscall records or Sysmon EventID 1 logs showing the agent process (openclaw, python agent.py, etc.) as parent of unexpected child processes such as bash, sh, curl, wget, or git — these represent the execution leg of a successful agentjacking where injected instructions caused the agent to invoke system tools. | Secrets manager access logs scoped to the agent's IAM role or Vault token: AWS CloudTrail 'GetSecretValue' events, HashiCorp Vault audit log entries for the agent token, or .env file access timestamps (stat ~/.env, stat ./.secrets) — anomalous access timestamps or access to secrets paths outside the agent's documented task scope are high-confidence indicators of credential harvesting via injected instructions. | Outbound network connection logs from the agent host during the ingestion window: specifically any connections to destinations not in the agent's pre-documented API allowlist, captured via 'ss -tnp' snapshots, mitmproxy session logs, or firewall egress logs — an attacker embedding exfiltration instructions in a poisoned Sentry error event would cause the agent to initiate an outbound POST to an attacker-controlled endpoint carrying harvested secrets or code.

Per-Action IR Details

Step 1: Assess AI agent deployment — inventory every AI coding agent operating in your environment, including OpenClaw or similar agentic frameworks, and document what data sources each agent is permitted to ingest (logs, error-tracking platforms, dependency feeds, external APIs).

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: Establishing IR Capability and Asset Visibility

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

Compensating: Run 'ps aux | grep -E "openclaw|agent|langchain|copilot"' and 'systemctl list-units --type=service | grep -i agent' on all developer workstations and CI/CD runners to surface running agent processes. Use osquery query 'SELECT name, path, cmdline FROM processes WHERE cmdline LIKE "%agent%" OR cmdline LIKE "%openclaw%";' to enumerate agent instances. Document each agent's config file (e.g., openclaw.yaml or .agent/config.json) to extract permitted data source bindings including Sentry DSN endpoints, dependency registry URLs, and log file paths.

Evidence: This is a pre-incident inventory step that does not alter live state. No volatile capture is required before execution. However, snapshot all agent configuration files and environment variable listings (printenv | grep -iE 'SENTRY|API_KEY|TOKEN|SECRET') before any downstream remediation steps, as these establish the pre-remediation permission baseline.

Step 2: Apply least privilege to agent accounts — enforce NIST AC-6 (Least Privilege) and CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts) by scoping AI agent permissions to the minimum required for each task; remove standing access to secrets stores, CI/CD pipelines, and production systems where not operationally necessary.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment: Limiting Blast Radius of Compromised Agent Accounts

Controls: NIST AC-6 (Least Privilege), NIST AC-2 (Account Management), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: For GitHub Actions or GitLab CI: audit repository-level token scopes via 'gh api /repos/{owner}/{repo}/actions/secrets' and revoke any GITHUB_TOKEN with write or admin scope not strictly needed by the agent workflow. For HashiCorp Vault: run 'vault token lookup ' to enumerate policies, then 'vault token revoke ' to revoke standing secrets access; re-issue with a narrow policy scoped to a single path. For AWS: run 'aws iam simulate-principal-policy --policy-source-arn --action-names secretsmanager:GetSecretValue' to audit effective permissions, then remove via 'aws iam detach-role-policy'.

Evidence: Before revoking any agent tokens or modifying IAM/Vault policies, capture: (1) current agent session tokens and their creation timestamps from the secrets manager audit log (e.g., AWS CloudTrail 'GetSecretValue' events for the agent's IAM role over the past 30 days); (2) 'netstat -ano' or 'ss -tnp' output on the agent host to record active outbound connections the agent currently holds — an agentjacking scenario may have the agent maintaining a live connection to an attacker-controlled API endpoint that disappears once credentials are rotated; (3) running process tree of the agent ('ps -ef --forest' or Get-CimInstance Win32_Process) to detect any child processes spawned via injected instructions.

Step 3: Enforce MFA and access controls on developer tooling — require MFA on all systems AI agents can reach or authenticate against, per CIS 6.3 (Require MFA for Externally-Exposed Applications) and CIS 6.5 (Require MFA for Administrative Access); apply D3-MFA and D3-UAP (User Account Permissions) countermeasures to agent service accounts.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment: Hardening Access Paths Reachable by a Hijacked Agent

Controls: NIST AC-17 (Remote Access), NIST AC-3 (Access Enforcement), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access)

Compensating: AI agent service accounts are non-interactive and cannot complete MFA challenges; the compensating control is IP-allowlisting and short-lived token issuance in lieu of MFA. For GitHub: enable required OIDC-based token exchange in Actions workflows ('id-token: write' permission with audience scoping) to replace

long-lived PATs. For Sentry integrations: rotate the Sentry DSN and internal integration token, then restrict the new token's scope to 'event:read' only via Sentry's API token permission UI. For NPM/PyPI dependency feeds: configure '.npmrc' or 'pip.conf' to pin to internal mirror registries, removing direct agent access to public registries that could deliver poisoned package metadata.

Evidence: Before enforcing MFA policy changes or revoking existing authentication tokens on developer tooling, capture: (1) agent authentication logs from each target system (e.g., Sentry audit log export via 'GET /api/0/organizations/{org}/audit-logs/?event=member.invite' and equivalent token-use events) to establish whether an in-progress agentjacking session has already authenticated to downstream systems; (2) any active OAuth or OIDC sessions the agent holds, retrievable via 'curl -H "Authorization: Bearer " https://api.github.com/user' to confirm token validity before revocation.

Step 4: Audit log ingestion pipelines — review which external or third-party data sources (e.g., Sentry, dependency registries) feed into AI agent context windows; validate that content from these sources is treated as untrusted input, not as authoritative instructions, per NIST AC-4 (Information Flow Enforcement).

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: Identifying Injection-Susceptible Data Flows

Controls: NIST AC-4 (Information Flow Enforcement), NIST AC-20 (Use Of External Systems), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Grep OpenClaw or equivalent agent configuration files for log ingestion bindings: 'grep -rE "sentry|logfile|feed|ingest|context_source" ~/.openclaw/ /etc/openclaw/ ./agent_config/'. For each identified source, manually review a sample of the last 100 ingested records using 'tail -100 | grep -iE "ignore previous|system:|assistant:||\[INST\]|you are now|disregard"' to detect prompt injection payloads embedded in log content. For Sentry: export the last 50 error events via Sentry's REST API ('GET /api/0/projects/{org}/{project}/events/') and scan event titles, stack trace messages, and breadcrumb fields for instruction-bearing strings characteristic of log-poisoning attacks.

Evidence: This is an analysis step that reviews existing log data and does not alter live state. Before making any pipeline configuration changes that follow this audit, preserve a read-only export of: (1) the current Sentry project event feed (last 7 days) including raw event JSON with breadcrumbs and contexts fields intact; (2) agent context window logs if OpenClaw persists conversation/tool-call history to disk (commonly at '~/.openclaw/sessions/' or equivalent); (3) dependency manifest lock files (package-lock.json, poetry.lock, requirements.txt) in their current pre-remediation state to establish whether any recently resolved package version delivered a poisoned install script or README that entered the agent's context.

Step 5: Enable audit logging for AI agent actions — implement AU-2 (Event Logging) and AU-12 (Audit Record Generation) to capture agent-initiated file access, code execution, secret retrieval, and API calls as discrete auditable events; configure AU-6 (Audit Record Review, Analysis, and Reporting) to alert on anomalous agent behavior.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: Establishing Behavioral Baselines for AI Agent Activity

Controls: NIST AU-2 (Event Logging), NIST AU-12 (Audit Record Generation), NIST AU-6 (Audit Record Review, Analysis, And Reporting), CIS 8.2 (Collect Audit Logs)

Compensating: Deploy Sysmon on Windows developer workstations running AI agents: configure EventID 1 (Process Create) to capture agent process spawning child shells (cmd.exe, powershell.exe, bash) and EventID 11 (File Create) for writes to sensitive directories (secrets stores, .ssh/, .aws/credentials). On Linux, enable auditd with rules: '-a always,exit -F arch=b64 -S execve -F ppid= -k agentjack_exec' and '-a always,exit -F arch=b64 -S openat -F path=/home//.ssh -k agentjack_cred_access'. For API call visibility with no SIEM, run the agent behind a local mitmproxy instance ('mitmproxy --mode transparent --ssl-insecure') to capture all outbound HTTPS calls the agent makes and flag any calls to non-whitelisted endpoints that may represent exfiltration initiated by injected instructions.

Evidence: Enabling logging does not alter the agent's live operational state. However, if the agent is currently running and suspected of compromise, capture before touching the process: (1) in-memory context window state if accessible via OpenClaw's debug API or equivalent ('curl http://localhost:/debug/context' or equivalent introspection endpoint); (2)

current file descriptor table ('ls -la /proc//fd') to identify any open file handles the agent holds on secrets files or CI/CD credential stores; (3) agent's outbound DNS queries via 'cat /proc//net/dns' or local resolver logs to detect any novel domains contacted as a result of injected instructions prior to logging being enabled.

Step 6: Update threat model — add indirect prompt injection via log poisoning as a named attack pattern in your threat register, mapped to T1059 and T1552, and assess whether existing detection rules cover agent-driven command execution and credential access.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Lessons Learned and Threat Model Improvement

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

Compensating: Write YARA rules targeting prompt injection payloads in log content: rule strings should match patterns like 'ignore previous instructions', 'system:', '', and OpenClaw-specific tool-call syntax that should never appear in error log fields. Store rules in a shared Git repo and run via 'yara -r agentjack.yar /var/log/sentry_export/' on scheduled cron. Author Sigma rules for auditd/Sysmon targeting agent process trees spawning unexpected child processes: condition on 'ParentImage contains openclaw AND Image in (cmd.exe, powershell.exe, /bin/sh, /bin/bash, curl, wget)' to catch command execution triggered by injected log instructions. Test rules against the Sentry event export captured during Step 4.

Evidence: This is a post-incident documentation and engineering step; it does not alter live system state. No volatile capture is required. Input artifacts for threat model update should include: the preserved Sentry event exports and agent session logs from Steps 4 and 5, any identified injection payloads found during the audit, and a record of which CI/CD pipelines and secrets stores were within the agent's blast radius as documented in Step 1.

Step 7: Monitor for follow-up research and vendor response — track updates to the arXiv taxonomy (2603.27517), Eye Security blog, and reco.ai disclosures; watch for patches or architectural guidance from AI agent framework maintainers including OpenClaw.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Ongoing Intelligence Integration and Vendor Coordination

Controls: CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Configure RSS or Atom feed monitoring for the OpenClaw GitHub releases page ('https://github.com//openclaw/releases.atom' — validate this URL manually) using a free tool such as RSS-Bridge or a cron-driven 'curl | grep -i "patch\|security\|advisory"' script. Subscribe to the CISA Known Exploited Vulnerabilities (KEV) feed (https://www.cisa.gov/sites/default/files/feeds/known_exploited_vulnerabilities.json) and filter for AI agent and LLM-related entries. For arXiv, set a Google Scholar alert on '2603.27517' and 'agentjacking prompt injection' to catch follow-on academic disclosures. Log all vendor advisory receipts and response timelines in the threat register updated in Step 6.

Evidence: This is a sustained monitoring and intelligence step that does not alter live state. No volatile capture is required. Maintain a running evidence package that includes: version hashes of currently deployed OpenClaw binaries ('sha256sum ') to enable rapid comparison when a vendor patch is released; a snapshot of current agent dependency lock files to diff against post-patch state; and a dated record of the pre-patch configuration baseline from Step 1 to support post-patch regression testing.

Detection Guidance

Detection for Agentjacking requires monitoring at the agent behavior layer, not just the network perimeter. Key signals to hunt for:

1. Agent-initiated process execution: Alert on any process spawned by an AI agent process that was not explicitly invoked by a human operator, especially shell interpreters, scripting engines, or credential-access

utilities. Reference MITRE T1059 and apply system file analysis to monitor agent working directories for unexpected file writes or modifications.

2. Credential and secrets access: Monitor for AI agent processes querying environment variables, secrets managers, or credential stores outside of documented workflows. This maps to T1552 (Unsecured Credentials) and warrants credential rotation as a compensating control if access cannot be denied.

3. Anomalous log ingestion volume or source: Alert on AI agents ingesting logs or error-tracking data from sources not in an approved allowlist, particularly new Sentry projects, unfamiliar dependency feeds, or log files modified by external processes immediately before agent access.

4. Lateral movement from agent accounts: Track authentication events and API calls originating from AI agent service accounts against systems outside their normal operational scope. Cross-reference with NIST AU-2 event logging for CI/CD pipeline access, repository writes, and external API calls.

5. Startup configuration changes: Apply system init configuration analysis to detect modifications to agent configuration files, plugin manifests, or tool-use permissions that could expand the agent's ingestion surface without explicit operator approval.

Log sources to prioritize: agent process logs, secrets manager access logs, CI/CD audit trails, Sentry integration event logs, and any orchestration layer managing agent task queues. Organizations without dedicated AI agent audit logging should treat its absence as a critical gap requiring immediate remediation under NIST AU-12.

Indicators of Compromise

Type	Value	Context	Confidence
TOOL	`Pending — refer to Eye Security blog (eye.security/blog/log-poisoning-openclaw-ai-agent-injection-risk) for published indicators`	Eye Security's log poisoning analysis of OpenClaw may contain specific payload patterns, injection strings, or behavioral indicators; values were not reproduced in the source material available for this report	LOW
TOOL	`Pending — refer to arXiv taxonomy (arxiv.org/html/2603.27517v1) for documented attack patterns`	The systematic taxonomy published on arXiv documents specific injection techniques and case studies for the OpenClaw framework; concrete IOC values were not extractable from the provided source text	LOW

Framework Mappings

MITRE-ATTACK

- **T1190** — Exploit Public-Facing Application
- **T1059** — Command and Scripting Interpreter
- **T1574** — Hijack Execution Flow
- **T1552** — Unsecured Credentials

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
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-10** — Information Input Validation
- **AT-2** — Literacy Training and Awareness

OWASP-TOP10-2021

- **A03:2021** — Injection

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **8.2** — Collect Audit Logs

ISO-27001-2022

- **A.8.26** — Application security requirements

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
T1059	Command and Scripting Interpreter	Execution
T1574	Hijack Execution Flow	Persistence
T1552	Unsecured Credentials	Credential-Access

Sources

Source	URL	Tier
OpenClaw: The AI Agent Security Crisis Unfolding Right Now	https://www.reco.ai/blog/openclaw-the-ai-agent-security-crisis-unfo...	T3

Source	URL	Tier
Sentry: Application Performance Monitoring & Error Tracking Software	https://sentry.io/	T3
Log poisoning in AI agents: The OpenClaw case - Eye Security	https://www.eye.security/blog/log-poisoning-openclaw-ai-agent-injec...	T3
OpenClaw security is worse than I expected and I'm not sure what to ...	https://www.reddit.com/r/AI_Agents/comments/1r3u98p/openclaw_securi...	T3
A Systematic Taxonomy of Security Vulnerabilities in the OpenClaw ...	https://arxiv.org/html/2603.27517v1	T2

Generated 2026-06-14 04:25 UTC by TJS Security Command Center