INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-07-01 07:10 UTC

Shell Injection Class Flaw Bypasses Safety Checks in 10 of 11 AI Coding Agents, Exposing Credentials and CI/CD Pipelines

SECURITY ANALYSIS | HIGH | CVSS 7.5

SCC Item ID	SCC-STY-2026-0311
Type	Security Analysis
Severity	HIGH
CVSS Base Score	7.5
Affected Products	opencode, Goose, Cline, Roo-Code, Aider, Plandex, Open Interpreter, OpenHands, SWE-agent, Hermes (all vulnerable); Claude Sonnet 4.6 via affected agents; Continue (not affected in default mode), no specific versions confirmed at time of publication
Published	2026-06-30T10:26:15
Discovery Source	Rss

Executive Summary

Adversa AI's GuardFall research, reported by The Hacker News, found that 10 of 11 tested open-source AI coding agents, including opencode, Goose, Cline, Roo-Code, Aider, Plandex, Open Interpreter, OpenHands, SWE-agent, and Hermes, can be manipulated into executing malicious shell commands despite built-in safety filters. The core flaw is architectural: agents evaluate a raw command string for safety, but bash executes a semantically rewritten version that bypasses the check entirely, a gap that maps to decades-old injection vulnerability classes. The highest-risk exposure is agents running in auto-execute mode inside CI/CD pipelines, where a successful bypass can silently exfiltrate SSH keys, cloud credentials, and any secrets accessible to the agent's runtime account, placing software supply chains and cloud environments at direct risk.

Technical Analysis

Adversa AI's GuardFall research presents a vulnerability class, not a single discrete flaw. The underlying mechanism is a semantic gap between safety inspection and actual execution. AI coding agents that permit shell command execution typically implement text-based filtering: the agent inspects a command string, determines it appears safe, and proceeds with execution. The flaw is that bash and similar shells do not execute the literal string the filter inspected, they parse and rewrite it according to shell grammar, including quote handling, variable expansion, command substitution, and encoding interpretation. Classic bash quoting

techniques and obfuscation patterns, some dating to the 1970s Unix era, produce a command string that passes the filter but executes a semantically different payload.

This maps directly to CWE-78 (OS Command Injection), CWE-77 (Improper Neutralization of Special Elements used in a Command), and CWE-116 (Improper Encoding or Escaping of Output), with CWE-184 (Incomplete List of Disallowed Inputs) and CWE-20 (Improper Input Validation) characterizing the filter design failures. The researchers tested 11 open-source agents; 10 were vulnerable. Only Continue, in its default mode, was reported not affected.

The MITRE ATT&CK techniques relevant to successful exploitation include T1059.004 (Unix Shell) and T1059.007 (JavaScript) for execution, T1552.001 (Credentials in Files), T1552.004 (Private Keys), and T1552.005 (Cloud Instance Metadata API) for credential access, T1027.010 (Command Obfuscation) and T1140 (Deobfuscate/Decode Files or Information) for the obfuscation layer, T1190 (Exploit Public-Facing Application) and T1195.001 (Compromise Software Dependencies and Development Tools) for the supply chain angle, and T1204.002 (Malicious File) for user-execution scenarios where a developer opens a malicious project.

The highest-risk deployment scenario is agentic automation inside CI/CD pipelines. When an agent runs in auto-execute mode, no human confirmation required before shell commands run, an injected payload can operate entirely without user interaction. A developer cloning a repository containing a malicious prompt or configuration file could trigger credential exfiltration before any manual review occurs. The affected agents collectively represent approximately 548,000 GitHub stars, according to Adversa AI as reported by The Hacker News, indicating broad adoption across development teams.

Important sourcing note: the primary source for all claims in this story is a single Tier 2 publication (The Hacker News) reporting on Adversa AI's own research. Independent corroboration from NVD, CISA, or vendor security advisories had not been confirmed at time of analysis. No CVE identifiers have been assigned and no vendor patches were confirmed. Core claims should be treated as credible but not yet independently verified.

Action Checklist

1. Step 1: Assess exposure, audit your development environment for any of the ten affected agents: opencode, Goose, Cline, Roo-Code, Aider, Plandex, Open Interpreter, OpenHands, SWE-agent, and Hermes; document which teams use them and in what modes
2. Step 2: Disable or restrict auto-execute mode, immediately configure affected agents to require human confirmation before any shell command executes; treat auto-execute mode in CI/CD pipelines as unacceptable risk until patches are confirmed; reference NIST AC-3 (Access Enforcement) and AC-6 (Least Privilege) as the governing controls
3. Step 3: Audit CI/CD pipeline permissions, apply least-privilege principles (NIST AC-6, CIS 5.4) so agents running in pipelines operate under accounts with only the minimum permissions required; revoke access to SSH key stores, cloud credential files, and secrets managers that agent accounts do not operationally require
4. Step 4: Rotate potentially exposed credentials, treat any SSH keys (T1552.004), cloud credentials (T1552.005), and secrets-in-files (T1552.001) accessible to agent runtime accounts as potentially compromised; rotate per your credential rotation procedures (NIST IA-5) and audit access logs for anomalous exfiltration activity

5. Step 5: Review repository intake controls, establish or enforce policies restricting which external repositories developers may open directly inside an agentic coding session; untrusted repository content is a plausible injection vector (T1195.001, T1204.002)
6. Step 6: Update threat model, add AI coding agent shell-injection as an explicit attack vector in your software supply chain threat register; map to T1059.004, T1195.001, and T1552 sub-techniques and assign ownership for monitoring
7. Step 7: Monitor for vendor patches and authoritative advisories, track Adversa AI, NVD, and the GitHub repositories of each affected project for patch releases, CVE assignments, or CISA advisories; no patches were confirmed at time of publication
8. Step 8: Brief engineering leadership, communicate organizational exposure with specific agent names, pipeline locations, and credential risk context; request confirmation that auto-execute restrictions have been applied before next CI/CD pipeline run

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate immediately to CISO and legal counsel if CloudTrail, SSH auth logs, or secrets-manager access logs show any API calls to GetSecretValue, AssumeRole, or SSH authentication events using agent-runtime credentials outside of expected pipeline execution windows, as this indicates credential exfiltration has moved from theoretical to confirmed and may trigger breach notification obligations under applicable data protection regulations.
Recovery Notes	After containment (auto-execute disabled) and eradication (credentials rotated, permissions reduced), restore CI/CD pipeline agent invocations only for agents where the vendor has issued a confirmed patch addressing the safety-check bypass architecture — not merely a configuration workaround. Before re-enabling any agent in a pipeline, validate that the fix enforces confirmation on the semantically executed command string, not the pre-rewrite string evaluated by the safety filter, as the GuardFall flaw is architectural. Monitor agent parent-child process trees via Sysmon Event ID 1 (Process Create) and shell history logs for a minimum of 30 days post-recovery, alerting on any shell process (bash, sh, zsh, cmd.exe, powershell.exe) spawned by an agent process outside of an approved command allowlist.

Forensic Artifacts	Agent session transcript and chat history files specific to each affected tool — e.g., `.aider.chat.history.md` (Aider), `~/.config/goose/sessions/*.jsonl` (Goose), Cline VSCode extension logs in `%APPDATA%/Code/logs/` — which record the raw LLM-generated command strings before and after any safety-check rewrite, directly evidencing the bypass mechanism | Shell history files (`~/.bash_history`, `~/.zsh_history`) for the user account under which the agent ran, filtered for commands executed during agent session windows — these capture the semantically rewritten commands that bash actually executed, distinct from what the safety filter evaluated | CI/CD runner environment variable dumps and secrets-manager access logs (AWS CloudTrail `GetSecretValue`, HashiCorp Vault audit log `secret/data/*` read events, GitHub Actions `GITHUB_TOKEN` usage in workflow run logs) timestamped to agent execution windows, evidencing whether credential access occurred | Repository-level files that serve as the injection vector for this GuardFall vulnerability class: `.promptfiles`, `AGENTS.md`, `CLAUDE.md`, `AGENT_INSTRUCTIONS.md`, and any Makefile or package.json `scripts` blocks in repositories opened during agent sessions — preserve these at the exact commit hash present during the session | Process creation logs from Sysmon Event ID 1 (Windows) or Linux auditd `execve` syscall records filtered on parent processes matching agent binary names (aider, goose, opencode, interpreter, openhands, swe-agent, plandex, hermes), capturing child shell processes and their full command-line arguments as executed by the OS, which represents ground truth on what commands the safety-check bypass actually ran

Per-Action IR Details

Step 1: Assess exposure — audit your development environment for any of the ten affected agents: opencode, Goose, Cline, Roo-Code, Aider, Plandex, Open Interpreter, OpenHands, SWE-agent, and Hermes; document which teams use them and in what modes

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 — Preparation: Establishing IR capability requires knowing which systems and tools are in scope before an incident occurs

Controls: CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Run `pip list`, `npm list -g`, and `which` checks across developer workstations and CI/CD runner nodes to identify installed agent binaries (e.g., `which aider`, `which opencode`, `which goose`). Query package managers: `pip show open-interpreter cline roo-code plandex openhands swe-agent hermes 2>/dev/null`. On Windows runners, use `Get-Command aider,goose,opencode -ErrorAction SilentlyContinue`. Document output in a shared spreadsheet with team name, host, agent version, and whether auto-execute mode is enabled.

Evidence: This step does not alter live state. However, before engaging developers for survey responses, snapshot the current state of CI/CD runner environment variables (`printenv | grep -iE 'OPENAI|ANTHROPIC|AWS|AZURE|GCP|SECRET|KEY|TOKEN'`) and agent configuration files (e.g., `~/.aider.conf.yml`, `~/.config/opencode/`, `~/.cline/settings.json`) in case agents are already running in a compromised state. These configs may reveal which credential stores the agents have access to.

Step 2: Disable or restrict auto-execute mode — immediately configure affected agents to require human confirmation before any shell command executes; treat auto-execute mode in CI/CD pipelines as unacceptable risk until patches are confirmed; reference NIST AC-3 (Access Enforcement) and AC-6 (Least Privilege) as the governing controls

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: Short-term containment stops the attack from spreading while preserving the ability to investigate

Controls: NIST AC-3 (Access Enforcement), NIST AC-6 (Least Privilege)

Compensating: For agents without a built-in confirmation flag, wrap the agent invocation in a shell script that intercepts command execution and prompts `read -p 'Execute? [y/N]: ' confirm` before passing through. For CI/CD pipelines (GitHub Actions, GitLab CI, Jenkins), set the agent invocation step to `dry-run` or `--no-auto-run` mode where the flag exists (e.g., `aider --no-auto-run`, `open-interpreter --safe-mode`). Where no such flag exists, disable the pipeline stage entirely until vendor guidance is available.

Evidence: BEFORE disabling auto-execute or altering agent configuration in any environment where the agent has already run: capture the shell history of the agent's runtime user (`~/.bash_history`, `~/.zsh_history`), the agent's session log files (e.g., `.aider.chat.history.md` for Aider, Goose session transcripts in `~/.config/goose/sessions/`), and any recently created or modified files in the working directory (`find . -newer /tmp/baseline -type f -ls`). This captures evidence of any commands the agent may have already executed via the bypass mechanism before containment.

Step 3: Audit CI/CD pipeline permissions — apply least-privilege principles (NIST AC-6, CIS 5.4) so agents running in pipelines operate under accounts with only the minimum permissions required; revoke access to SSH key stores, cloud credential files, and secrets managers that agent accounts do not operationally require

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: Limiting the blast radius of a potential compromise by restricting what the attacker can reach through the compromised agent

Controls: NIST AC-6 (Least Privilege), NIST AC-3 (Access Enforcement), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Enumerate permissions of the CI/CD service account running the agent: `id `, `sudo -l -U `, `cat /etc/sudoers.d/*`. List accessible secrets: check for `~/.aws/credentials`, `~/.ssh/id_*`, `/run/secrets/`, `$GITHUB_TOKEN`, `$CI_JOB_TOKEN` in the runner environment. Revoke AWS IAM policy attachments using `aws iam list-attached-user-policies --user-name ` and detach non-essential policies. For GitHub Actions, audit and narrow `permissions:` blocks in workflow YAML files to the minimum required scopes.

Evidence: BEFORE revoking any permissions or access tokens: capture the current IAM/RBAC state (`aws iam get-user`, `gcloud iam service-accounts get-iam-policy`, `kubectl auth can-i --list --as=`), active session tokens (`aws sts get-caller-identity`), and CloudTrail/audit logs for the agent's service account covering the prior 30 days filtered on `CreateKey`, `GetSecretValue`, `AssumeRole`, and `SSHPublicKey` API calls. These establish a baseline of what the agent account could access and whether that access was used anomalously before the permission change destroys the live state.

Step 4: Rotate potentially exposed credentials — treat any SSH keys (T1552.004), cloud credentials (T1552.005), and secrets-in-files (T1552.001) accessible to agent runtime accounts as potentially compromised; rotate per your credential rotation procedures (D3-CRO) and audit access logs for anomalous exfiltration activity

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: Removing threat artifacts and neutralizing attacker-controlled access by invalidating credentials the agent may have exfiltrated

Controls: NIST AC-2 (Account Management), NIST IA-5 (Authenticator Management) — note: IA-5 governs authenticator lifecycle including rotation; cited from NIST 800-53r5 IA family per knowledge base scope, CIS 5.2 (Use Unique Passwords), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Use `aws iam list-access-keys --user-name ` to identify and then rotate or delete keys with `aws iam delete-access-key`. Revoke and regenerate SSH keys: `ssh-keygen -R ` and remove public keys from `~/.ssh/authorized_keys` on target hosts. For GitHub, use `gh auth status` and revoke tokens via the GitHub UI or API (`gh api -X DELETE /user/keys/`). Search the codebase and CI/CD configs for hardcoded secrets using truffleHog: `trufflehog filesystem --directory=. --only-verified`. Log all rotations with timestamps for the post-incident record.

Evidence: BEFORE rotating any credential: pull the complete access log for that credential. For AWS: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue= --start-time ` filtered on `GetSecretValue`, `ListBuckets`, `DescribeInstances`, `CreateNetworkInterface`. For SSH keys: review `auth.log` or `/var/log/secure` for successful authentications using the key fingerprint (`ssh-keygen -lf ~/.ssh/id_rsa.pub`). For secrets-in-files: check git log for any commits that may have pushed credential files upstream (`git log --all --full-history -- '*.env' '*.pem' '*.key'`).

Rotation invalidates the audit trail linkage between the credential and the access events — capture first.

Step 5: Review repository intake controls — establish or enforce policies restricting which external repositories developers may open directly inside an agentic coding session; untrusted repository content is a plausible injection vector (T1195.001, T1204.002)

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: Removing the conditions that allowed exploitation, including the intake pathway through which malicious content could reach the agent's safety-check bypass

Controls: NIST AC-4 (Information Flow Enforcement), CIS 2.3 (Address Unauthorized Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Implement a pre-clone hook via `git config --global core.hooksPath` pointing to a script that checks the remote URL against an allowlist before permitting clone into an agent working directory. For GitHub organizations, use branch protection rules and CODEOWNERS to prevent agents from being invoked on PRs from forked repositories. Document an approved-repositories policy and distribute via developer onboarding docs. As a quick manual control, require developers to run `git log --oneline -20` and inspect `.github/workflows/`, `Makefile`, `pyproject.toml`, and `package.json` for suspicious script hooks before opening any external repo in an agent session.

Evidence: This step is primarily policy-setting and does not alter live state. However, if a specific malicious repository is suspected as the injection source, preserve its content before any network or access controls are applied: archive the repository at the suspected malicious commit hash (`git archive --format=tar HEAD > /tmp/suspicious-repo-.tar`), capture the git reflog (`git reflog show --all`), and preserve any `.promptfiles`, `AGENTS.md`, `CLAUDE.md`, or similar agent-instruction files in the repository root that are the documented injection mechanism for this GuardFall vulnerability class.

Step 6: Update threat model — add AI coding agent shell-injection as an explicit attack vector in your software supply chain threat register; map to T1059.004, T1195.001, and T1552 sub-techniques and assign ownership for monitoring

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Lessons learned and threat model updates prevent recurrence and improve detection for the next wave of AI agent vulnerabilities

Controls: NIST RA-3 (Risk Assessment) — note: RA-3 governs threat and risk identification; cited from NIST 800-53r5 RA family per knowledge base scope, CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Add a threat entry to your risk register (spreadsheet or JIRA) titled 'AI Coding Agent Shell-Injection via Safety-Check Bypass (GuardFall class)' with fields: attack vector (malicious content in repository files or LLM prompts), affected assets (developer workstations and CI/CD runners with opencode/Goose/Cline/Roo-Code/Aider/Plandex/Open Interpreter/OpenHands/SWE-agent/Hermes installed), likelihood (HIGH — no patch confirmed), impact (credential theft, CI/CD pipeline compromise), owner (AppSec or DevSecOps lead), and review date (30 days). Assign a Sysmon rule to alert on shell processes spawned by agent parent processes.

Evidence: No live-state alteration in this step. Collect and archive as supporting evidence for the threat model entry: the Adversa AI GuardFall research paper URL, screenshots of which agents are installed per the Step 1 audit, the CI/CD permission audit results from Step 3, and any anomalous access log entries identified during credential review in Step 4. These form the evidentiary basis for the risk rating assigned to this new threat entry.

Step 7: Monitor for vendor patches and authoritative advisories — track Adversa AI, NVD, and the GitHub repositories of each affected project for patch releases, CVE assignments, or CISA advisories; no patches were confirmed at time of publication

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Continuous monitoring for vendor remediation and authoritative guidance is required when no patch exists at time of incident declaration

Controls: CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), NIST SI-5 (Security Alerts, Advisories, and Directives) — note: SI-5 governs receipt and dissemination of security advisories; cited from NIST 800-53r5 SI family per knowledge base scope

Compensating: Set up GitHub repository watch notifications (Watch → Custom → Releases) on the GitHub repos for each of the ten affected agents: `opencode-ai/opencode`, `block/goose`, `cline/cline`, `RooVetGit/Roo-Code`, `Aider-AI/aider`, `plandex-ai/plandex`, `OpenInterpreter/open-interpreter`, `All-Hands-AI/OpenHands`, `SWE-agent/SWE-agent`, and the Hermes project repo. Subscribe to NVD CVE feed filtered by keyword 'AI agent' via RSS (`https://nvd.nist.gov/feeds/json/cve/1.1/nvdcve-1.1-recent.json.gz`). Add CISA Known Exploited Vulnerabilities catalog RSS feed to a shared Slack channel.

Evidence: No live-state alteration. Document the monitoring setup itself as an artifact: record the GitHub watch confirmations, RSS feed subscriptions, and assigned owner (by name) in the incident ticket. Note the publication date of the Adversa AI GuardFall research as the baseline — any vendor commit referencing 'GuardFall', 'shell injection', 'safety bypass', or 'command confirmation' in the affected repos after this date is a patch candidate requiring immediate validation and re-triage.

Step 8: Brief engineering leadership — communicate organizational exposure with specific agent names, pipeline locations, and credential risk context; request confirmation that auto-execute restrictions have been applied before next CI/CD pipeline run

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Communicating incident findings and control decisions to leadership is required for organizational accountability and resource authorization

Controls: NIST IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting)

Compensating: Prepare a one-page brief using the audit outputs from Steps 1–3: list each affected agent by name, the CI/CD pipeline job names where it runs (from workflow YAML files), the credential types within reach (AWS keys, SSH keys, GitHub tokens), and the containment actions already taken (auto-execute disabled Y/N per pipeline). Include a binary confirmation checkbox for each pipeline owner: 'Auto-execute restricted before next run: [ ] Yes [ ] No — Owner signature/date.' Distribute via encrypted email or internal wiki with edit-locked version history to preserve the record.

Evidence: No live-state alteration. Attach to the leadership brief as appendices: the agent inventory from Step 1, the CI/CD permission audit summary from Step 3, and the credential rotation log from Step 4 (with timestamps but without the credential values themselves). These artifacts substantiate the exposure claims and provide the factual basis for any downstream regulatory disclosure assessment if credentials are confirmed exfiltrated.

Detection Guidance

No verifiable IOC values (hashes, domains, IPs) were published in the available source material. The cited source (Adversa AI via The Hacker News) may publish technical indicators in accompanying research artifacts, consult the Adversa AI GuardFall publication directly for payload samples or detection signatures.

Behavioral hunting priorities based on the attack class:

Shell obfuscation patterns: Hunt for processes spawned by AI agent runtimes (e.g., Python interpreter processes associated with Open Interpreter, Node.js processes associated with Cline or Roo-Code) that execute shell commands containing unusual quoting, ANSI-C quoting ($'...'), base64-encoded substrings, or command substitution constructs. These are characteristic of the bypass techniques described in GuardFall. Reference NIST AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting), confirm shell execution events are captured at the endpoint and forwarded to your SIEM.

Credential file access from agent processes: Alert on any process associated with an AI coding agent reading files in ~/.ssh/, ~/.aws/credentials, ~/.config/gcloud/, environment variable stores, or CI/CD secrets directories.

This maps to T1552.001 and T1552.004. Use NIST SI-7 (Software, Firmware, and Information Integrity Monitoring) as the countermeasure framing, monitor file access events on sensitive credential paths.

Cloud metadata API queries from unexpected processes: Alert on HTTP requests to instance metadata endpoints (169.254.169.254 or equivalent) originating from AI agent processes. This maps to T1552.005.

Data exfiltration from pipeline accounts: Review outbound network connections from CI/CD runner accounts for unexpected destinations, particularly during or immediately after agent-assisted build steps. Anomalous DNS queries or HTTP POST activity from pipeline runners warrants investigation.

Local account monitoring: Apply NIST AC-2 (Account Management) to agent runtime accounts; flag any privilege escalation attempts, new SSH key generation, or credential file modifications associated with those accounts.

Log sources to prioritize: endpoint process execution logs (auditd or equivalent on Linux), file integrity monitoring on credential directories, CI/CD pipeline execution logs with command-level detail, outbound network flow logs from build runners, and cloud provider credential usage logs (AWS CloudTrail, GCP Audit Logs, Azure Monitor).

Framework Mappings

MITRE-ATTACK

- **T1552.005** — Cloud Instance Metadata API
- **T1190** — Exploit Public-Facing Application
- **T1204.002** — Malicious File
- **T1027.010** — Command Obfuscation
- **T1059.007** — JavaScript
- **T1140** — Deobfuscate/Decode Files or Information
- **T1059.004** — Unix Shell
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.001** — Credentials In Files
- **T1552.004** — Private Keys

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

OWASP-TOP10-2021

- **A03:2021** — Injection

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **2.5** — Allowlist Authorized Software
- **6.3** — Require MFA for Externally-Exposed Applications
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

ISO-27001-2022

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

HIPAA-SECURITY

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

SOC2-TSC

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1552.005	Cloud Instance Metadata API	Credential-Access
T1190	Exploit Public-Facing Application	Initial-Access
T1204.002	Malicious File	Execution
T1027.010	Command Obfuscation	Defense-Evasion
T1059.007	JavaScript	Execution
T1140	Deobfuscate/Decode Files or Information	Defense-Evasion
T1059.004	Unix Shell	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1552.004	Private Keys	Credential-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/06/guardfall-exposes-open-source-ai-...	T2
[Bug]: Its not possible to use Opencode Zen #18140 - GitHub	https://github.com/NousResearch/hermes-agent/issues/18140	T3
This is speculative, but I suspect that if we dropped one of the latest ...	https://news.ycombinator.com/item?id=47640521	T3
Claude Code vs Codex CLI vs Aider vs OpenCode vs Pi vs Cursor	https://thoughts.jock.pl/p/ai-coding-harness-agents-2026	T3

Generated 2026-07-01 07:10 UTC by TJS Security Command Center