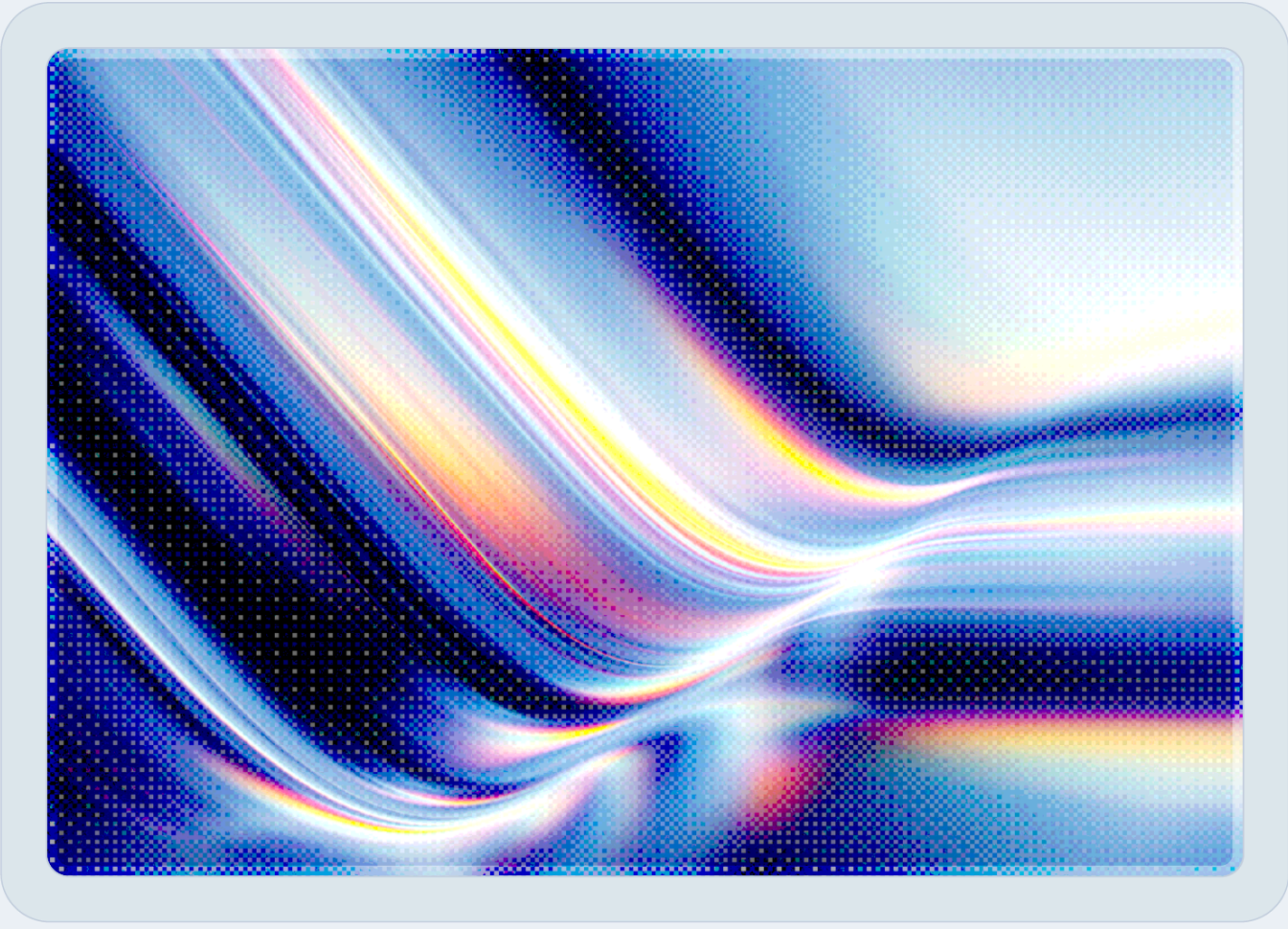
CSAI Foundation | Cloud Security Alliance

When the Model Never Runs: Agent Guardrail Bypasses

AWS, Google, and Vercel Agent Infrastructure Flaws Let Tool Calls Skip Model Authorization

2026-08-06

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

- Researchers disclosed a cross-platform vulnerability pattern, dubbed CoreBreak, showing that the tool-execution layers of Amazon Bedrock AgentCore, Google's Agent Development Kit (ADK), and Vercel's AI SDK harness packages could each be induced to run a tool without a legitimate model turn ever occurring [1].
- Because the underlying language model was never invoked, the model-level guardrails an organization typically relies on — system prompts, content filters, refusal training — were structurally irrelevant to the attack; there was no model decision left for those controls to intervene in [1].
- AWS assigned CVE-2026-18830 (CVSS v4.0 8.6) to the Bedrock AgentCore harness flaw, Google assigned CVE-2026-18236 (CVSS v4.0 9.3) to the ADK flaw, and Vercel's AI SDK packages received CVE-2026-64650 and CVE-2026-64651 (CVSS v4.0 6.3 each) [2][3][4][5][6].
- AWS's fix to the fully managed Bedrock AgentCore API deployed automatically and required no customer action. Google's and Vercel's fixes, by contrast, ship as package updates — ADK 2.5.0, and AI SDK harness-codex 1.0.29 and harness-opencode 1.0.28 — that self-hosted operators must apply themselves [1][2][4][5][6].
- The underlying design gap — trusting data "shaped like" a model-generated tool call without verifying it actually came from a model turn — is a structural pattern that CSA's prior GuardFall research identified in a different layer of the same agent stack, suggesting organizations should audit tool-dispatch trust boundaries broadly rather than treating this as three isolated bugs [8].

Background

Agentic AI systems built on frameworks such as Bedrock AgentCore, Google's ADK, and the harness packages distributed with Vercel's AI SDK follow a common architectural pattern. An orchestration layer, typically an SDK or runtime harness, assembles the user's request, the system prompt, prior conversation history, and a catalog of available tool definitions, then sends that bundle to a large language model. The model evaluates the request and, when it determines a tool is needed, returns a structured instruction naming the tool and its arguments. The SDK reads that instruction and dispatches the corresponding

function, script, or API call, then feeds the result back into the conversation for the model to incorporate into its response. Every guardrail an enterprise typically relies on — content filters, system-prompt instructions restricting sensitive actions, human-confirmation steps for high-risk tools, refusal training baked into the model itself — depends on the model actually being the party that decides whether and how a tool gets called.

Security researchers Hedi Ingber and Aviyam Ivgi, co-founders of the security firm Stealth, presented findings at Black Hat USA 2026 showing that this architectural assumption did not hold across three major agent platforms. Their research, given the name CoreBreak, demonstrated that the software components responsible for executing a model's tool-call decision — the harness or dispatch layer sitting between model output and actual tool execution — did not consistently verify that a tool-call instruction had legitimately originated from a model turn at all [1]. In each of the three cases the researchers examined, an attacker could get data that merely resembled a model-generated tool call in front of the dispatch logic, and the dispatch logic treated it as authoritative regardless of provenance. The practical consequence, as the researchers put it, is that "the model never ran at all," which means every guardrail layered on top of the model — no matter how well designed — never had an opportunity to evaluate the action before it executed [1].

This distinguishes the CoreBreak findings from the far more familiar category of prompt-injection attacks, where an adversary tries to manipulate a model into making an unsafe tool-call decision. Prompt injection targets the model's judgment; CoreBreak targets the plumbing that is supposed to enforce that judgment was ever exercised in the first place. An organization that has invested heavily in prompt-injection defenses, content moderation, and careful system-prompt design would gain little to no preventive protection against an attack that never touches the model's reasoning at all.

Security Analysis

Although the three vulnerabilities share a common conceptual root — insufficient verification of the provenance and authorization of a tool-call instruction before dispatch — each vendor's implementation created a distinct exploitation path, summarized in Table 1.

Vendor / Product	CVE	CVSS v4.0	Exploitation Path
AWS — Amazon Bedrock AgentCore	CVE-2026-18830	8.6 (High)	An authenticated remote caller could place a tool-use content block directly in the final message of an InvokeHarness API request; the

Vendor / Product	CVE	CVSS v4.0	Exploitation Path
(InvokeHarness API)			event loop dispatched the named tool without ever asking the model to authorize it [2][3].
Google – Agent Development Kit for Python	CVE-2026-18236	9.3 (Critical)	An attacker able to manipulate or inject events into an agent's session history could forge the human-approval confirmation for a sensitive tool, because the confirmation processor never checked that the target tool belonged to the executing agent, that it actually required confirmation, or that its name and arguments matched the original recorded call [4].
Vercel – @ai-sdk/harness-codex and @ai-sdk/harness-opencode	CVE-2026-64650 / CVE-2026-64651	6.3 (Medium) each	Malicious code already running inside a Linux sandbox could satisfy a process-path check that trusted any process whose command line contained the path of an approved helper script, allowing it to invoke host-exposed tools – including secret lookups, deployment operations, and cloud API calls – without a model-authorized event [5][6].

Table 1: Comparison of the three CoreBreak vulnerabilities disclosed for AWS, Google, and Vercel agent infrastructure.

The AWS flaw is the most directly reachable of the three: it required only an authenticated remote request, without any prerequisite compromise of a sandbox or session store. Google's flaw required either the ability to inject events into session history or to author function calls that the confirmation logic would treat as legitimate, which typically implies some existing foothold in the conversation pipeline, such as control over an upstream data source the agent ingests. Vercel's flaws sat one layer further back: they required untrusted code – a malicious dependency, build script, or lifecycle hook – already executing inside the sandbox before the process-path check could be abused to reach host-exposed tools. This prerequisite likely explains why their CVSS scores were markedly lower, since the CVSS attack-vector metric weighs the prerequisite of existing code execution against the same underlying design flaw of trusting an unverified signal as proof of model authorization [5][6].

The severity gradient across these three cases is a useful illustration for risk assessment: in this disclosure, at least, the impact of a missing-provenance-check flaw tracked how directly reachable the dispatch layer was from an untrusted actor, not merely how sensitive the tools behind it were. One plausible reason Google's ADK flaw scored highest is that forged confirmations could reach tools gated behind human approval, a control specifically intended to catch high-risk actions, while AWS's flaw likely scored high because it required no prior compromise at all, only a valid but otherwise unprivileged authenticated session. Vercel's exposure, by contrast, presupposed that an attacker had already achieved code execution inside the sandbox, which narrows — but does not eliminate — the practical attack surface, since sandbox environments that execute agent-generated code, third-party MCP tool outputs, or dependencies from public registries offer plausible injection points for that initial foothold. Organizations should treat this severity pattern as one data point from a single disclosure rather than a settled scoring rule for agent infrastructure generally.

A further implication concerns detection. In CSA's observation, security monitoring for agentic systems has generally focused on the model's inputs and outputs: logging prompts, flagging suspicious completions, and reviewing which tools a model chose to call and why — an emphasis consistent with CSA's prior research arguing that static, design-time guardrails are structurally insufficient on their own and must be paired with continuous, runtime verification [7]. When a tool executes without the model ever running, none of the artifacts that monitoring pipelines typically inspect — the model's chain of reasoning, its returned tool-call payload, its content-filter score — exist to be logged in the first place. Detecting this class of attack after the fact, or in real time, requires visibility into the dispatch and authorization layer itself: whether the tool-use instruction that reached the executor can be tied to an actual, corresponding model completion recorded in the session, not merely whether it looks well-formed.

Recommendations

Immediate Actions

Organizations running self-hosted or self-managed versions of the affected components should confirm patch status without delay: Google's ADK for Python fix shipped in version 2.5.0 on July 16, 2026, and Vercel's harness packages were fixed in version 1.0.29 for `@ai-sdk/harness-codex` and version 1.0.28 for `@ai-sdk/harness-opencode`, both released July 10, 2026 [3][4][5][6]. AWS's fix to the managed Bedrock AgentCore InvokeHarness API deployed automatically before July 31, 2026 and required no customer action, but organizations should still confirm through their AWS account activity or support channels that the fix applies to their region and configuration [2]. Security teams should also

review recent logs for any tool invocations on these platforms that cannot be tied to a corresponding, well-formed model completion in the session record, as a retrospective indicator of possible exploitation prior to patching.

Short-Term Mitigations

Teams building or operating agent infrastructure — whether on these three platforms or others with a similar SDK-to-model-to-tool architecture — should treat this disclosure as a prompt to audit their own dispatch layer for the same missing-provenance pattern: does the code that executes a tool call verify that the call actually originated from a model turn belonging to the correct agent and session, with arguments matching what the model returned, or does it merely check that the payload is shaped correctly? Where "human-in-the-loop" confirmation gates protect sensitive tools, teams should verify that the confirmation-processing logic checks tool ownership, confirmation requirement, and argument integrity against the original recorded call, rather than trusting a confirmation event's mere presence in the session history. For sandbox-based architectures resembling Vercel's, teams should replace any authorization check based on a process's command-line contents with a cryptographically verifiable signal, such as a signed, one-time authorization token issued specifically for that tool call.

Strategic Considerations

At a governance level, this disclosure argues for extending AI security assessments beyond the model itself to explicitly include the harness, SDK, and dispatch code that sits between model output and tool execution. Threat models that stop at "could the model be persuaded to call an unsafe tool" leave the far more direct question — "can the dispatch layer be reached without the model at all" — unexamined. Organizations procuring or building agentic AI platforms should ask vendors directly how tool-call authorization is verified end to end, and should favor architectures where every tool execution requires a signed, session-bound token tied to an actual model completion, rather than architectures that infer authorization from a message's structure or a process's identity.

CSA Resource Alignment

CSA's rapid research on GuardFall: Shell Injection Bypass Defeats AI Coding Agent Guardrails documented a structurally similar problem one layer removed from the CoreBreak findings: AI coding agents inspected raw command strings for unsafe patterns before the underlying shell performed its own expansion and substitution, creating a gap between what a guardrail evaluated and what actually executed [8]. CoreBreak's flaws sit at the layer directly above that one — the dispatch step that decides

whether a tool call is legitimate before any guardrail, shell-level or model-level, gets a chance to run at all. Read together, the two findings are early evidence that "guardrail bypass via structural gap between inspection and execution" may be a recurring pattern across the agentic AI stack rather than a one-off bug in any single vendor's product. Two disclosures across four vendors do not prove an industry-wide pattern, but they are enough to justify guard-architecture review — tokenize-then-verify rather than pattern-match-on-shape — as a standing practice rather than a one-time patch response.

CSA's research note on Legacy Infrastructure: The AI Agent Security Blind Spot makes a related argument at the identity and access layer: agentic AI security guidance has tended to focus on the model itself while underweighting the infrastructure an agent's tool calls actually touch [9]. CoreBreak reinforces that argument concretely — the AWS, Google, and Vercel flaws all demonstrate that the infrastructure enforcing (or failing to enforce) tool-call authorization is exactly the kind of non-model component that determines real-world security outcomes, and that infrastructure warrants the same scrutiny as the model's own alignment and filtering.

More broadly, these findings map directly onto CSA's MAESTRO agentic AI threat-modeling framework [10], particularly the layers addressing agent frameworks and deployment infrastructure, where MAESTRO explicitly calls for evaluating trust boundaries between an agent's reasoning component and its execution environment. Organizations conducting AICM-based assessments of agentic deployments should extend their AI Controls Matrix (AICM) v1.1 review [11] of execution-control and privilege-management domains to explicitly ask whether tool dispatch requires cryptographic proof of model authorization, not merely well-formed input, since CoreBreak demonstrates that this specific control gap existed in production code at three major cloud and platform vendors, disclosed together in a single coordinated research effort.

References

[1] The Hacker News. "AWS, Google, and Vercel Agent Flaws Let Attackers Trigger Tools Without Running the Model." The Hacker News, August 2026.

[2] AWS. "Security Bulletin: Amazon Bedrock AgentCore Harness Insufficient Input Validation (CVE-2026-18830)." Amazon Web Services, 2026.

[3] National Vulnerability Database. "CVE-2026-18830 Detail." NIST NVD, 2026.

[4] National Vulnerability Database. "CVE-2026-18236 Detail." NIST NVD, 2026.

[5] National Vulnerability Database. "CVE-2026-64650 Detail." NIST NVD, 2026.

[6] National Vulnerability Database. "CVE-2026-64651 Detail." NIST NVD, 2026.

[7] Cloud Security Alliance. "NIST Proves Static AI Guardrails Are Mathematically Insufficient." Cloud Security Alliance, June 2026.

[8] Cloud Security Alliance. "GuardFall: Shell Injection Bypass Defeats AI Coding Agent Guardrails." Cloud Security Alliance, July 2026.

[9] Cloud Security Alliance. "Legacy Infrastructure: The AI Agent Security Blind Spot." Cloud Security Alliance, June 2026.

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

[11] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2025.

© 2026 Cloud Security Alliance. AI-assisted Rapid Research.