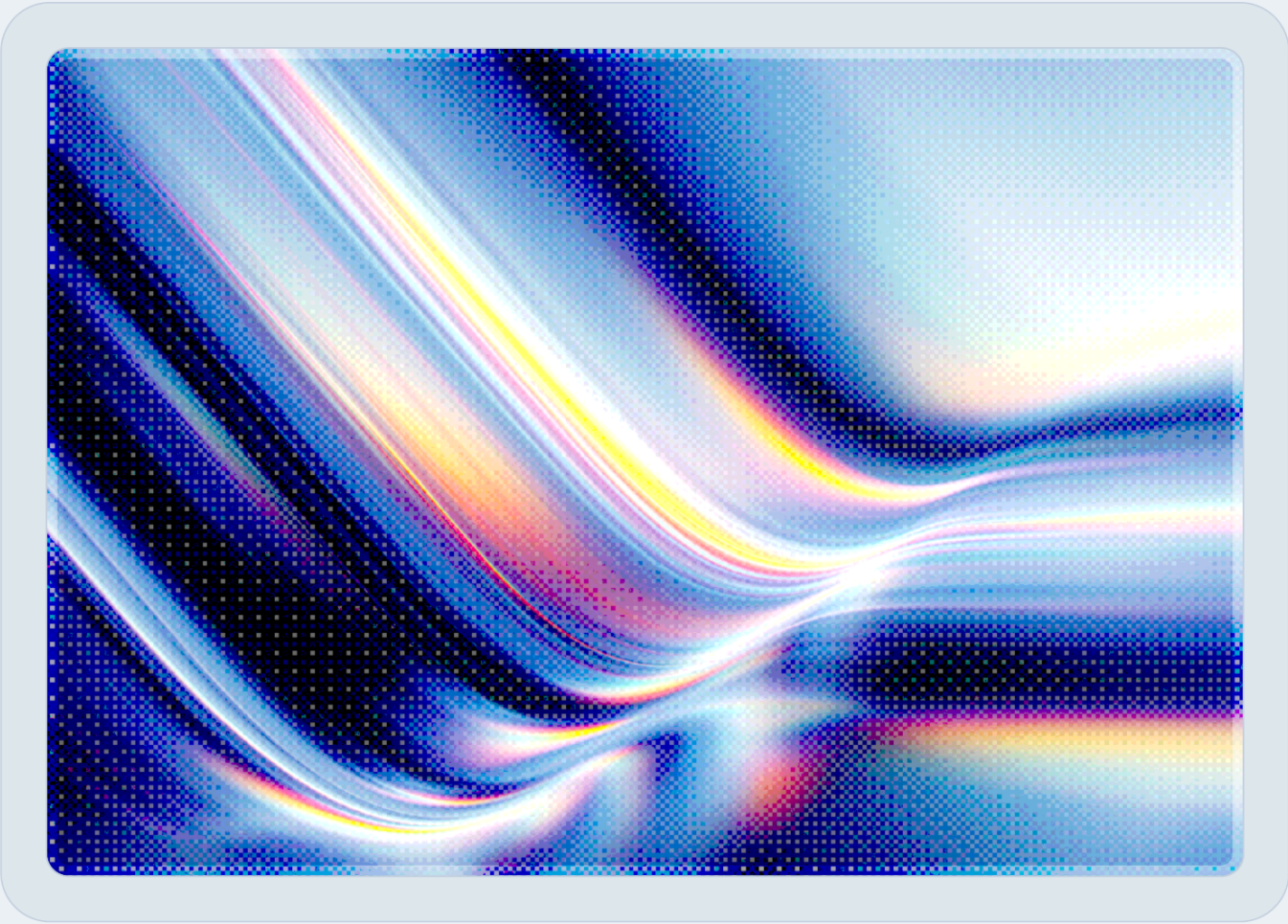
CSAI Foundation | Cloud Security Alliance

Google Deletes ADK Workflows After Agent-to-Agent Injection

2026-08-04

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Google removed three GitHub Actions workflows from its open-source Agent Development Kit (ADK) repository for Python after researchers at Pillar Security demonstrated that a public, low-privilege triage agent could be manipulated through prompt injection into triggering a maintainer-only agent with broad repository and cloud credentials [1][2][3].
- Pillar Security describes the finding as the first practical, real-world demonstration of agent-to-agent exploitation inside a production multi-agent system: rather than hijacking a single model, the attacker used one AI agent's output as the trigger for a second, more privileged agent [4].
- The attack chain worked because the privileged workflow validated *who* had posted a triggering comment rather than *whether the trusted account had itself been manipulated*, a gap that let a public GitHub issue or pull request indirectly author commands attributed to a trusted bot identity [1][4].
- A separate command allowlist meant to restrict the privileged agent to safe `git` and `gh` operations could be bypassed using `git`'s `core.hooksPath` and `alias` mechanisms, turning an apparently safe tool into a path to arbitrary code execution on the CI runner [4].
- The exposed secrets included a maintainer bot's personal access token — the credential actually exploited — a job-scoped `GITHUB_TOKEN` declared with write access to pull requests, issues, and repository contents but not itself used in the attack, and a Google Cloud service account key, illustrating how much authority modern CI/CD agent pipelines concentrate in a single automated identity [4].
- Google hardened the repository and deleted the affected workflows; its Vulnerability Rewards Program panel classified the underlying report as non-rewardable for a financial payout because exploitation required an element of social engineering, though Google did credit the researcher with an Honorable Mention. The technical mechanism — one agent manipulating another across a trust boundary — represents a structural weakness rather than a one-off bug [4].
- This disclosure follows Pillar's TrustIssues finding: a vulnerability in Google's gemini-cli rated CVSS 10.0, disclosed via a GitHub Security Advisory in April 2026 [7] and detailed further in Pillar's own write-up the following month, which described its reach into at least eight other repositories, including google/draco [6]. That earlier flaw let a single malicious GitHub issue

exfiltrate CI credentials and achieve full supply-chain compromise; the ADK case shows the same trusted-input weakness recurring in a genuinely multi-agent form.

Background

Google's Agent Development Kit (ADK) for Python is an open-source, code-first framework for building and orchestrating AI agents, distributed publicly on GitHub alongside companion sample and documentation repositories [5][8]. Like many actively maintained open-source AI projects, the ADK repository automated parts of its own maintenance using AI agents: one workflow triaged incoming issues and pull requests, and another, more privileged workflow could open fix commits or dismiss review requests once invoked by a maintainer. This pattern — delegating routine repository hygiene to an AI agent while reserving higher-impact actions for a smaller, trusted set of triggers — has become common across large open-source projects racing to keep pace with contribution volume. It also creates exactly the kind of internal trust boundary that security researchers have begun probing as agentic AI moves from single-agent assistants into multi-agent pipelines.

In early August 2026, Pillar Security researcher Dan Lisichkin published findings, subsequently covered by The Hacker News, The Register, and CSO Online, showing that the boundary between ADK's public-facing triage agent and its privileged maintenance agent could be crossed [1][2][3]. Google responded by deleting three workflows from the repository: an issue-analysis workflow that ran automatically whenever an issue was opened, an issue-fix workflow that listened for a `/adk-issue-fix` command, and a pull-request analysis workflow, alongside hardening of an earlier Gemini-based command-dispatch system that had the same underlying flaw [1]. Pillar frames the research, titled "I'll Just Call You: Agent-to-Agent Privilege Boundary Failures in CI/CD," as the first documented case of one production AI agent being weaponized against another rather than being exploited in isolation [4].

The disclosure is best understood in the context of Pillar's earlier work. Pillar's GitHub Security Advisory for "TrustIssues," published April 24, 2026, disclosed a vulnerability in Google's gemini-cli rated CVSS 10.0 [7]; the company's own write-up, published the following month, detailed how the same underlying flaw also affected at least eight other repositories, including google/draco, and could let a single malicious GitHub issue prompt-inject a Gemini-powered triage agent into exfiltrating workflow secrets and ultimately committing arbitrary code to a repository's main branch [6]. That vulnerability involved one compromised agent acting directly on stolen credentials. The ADK case extends the same underlying failure — an AI agent trusting content it should treat as adversarial — into a scenario where the compromised agent's *output* becomes the injection vector for a second, more privileged agent. Google

fixed the immediate technical issues in both cases, but the recurrence of the pattern across two separate repositories and disclosure cycles within the same year indicates a design gap that is likely to appear again as more organizations chain AI agents together in production automation.

Security Analysis

The ADK repository ran two generations of automation that shared the same structural weakness: a public, low-privilege agent and a maintainer-only, high-privilege agent connected by a comment-based trigger that neither system treated as an authorization boundary requiring independent verification. In the earlier Gemini-based system, the account controlling the privileged workflow, `adk-bot`, was in fact a human collaborator's personal access token rather than a dedicated non-human identity, meaning the token carried a real contributor's repository privileges [4]. An attacker could submit a pull request crafted to resemble a legitimate contribution while embedding a prompt injection payload; when the public triage agent processed it, the injected instructions caused the agent to post a comment beginning with `@gemini-cli`. Google's dispatch workflow treated that mention as a legitimate maintainer command and invoked the privileged workflow, which then held write access to pull requests and issues sufficient to edit comments, post fabricated approving reviews, and manufacture an apparently legitimate merge trail for poisoned code [4].

The newer Antigravity-based system, built around Google's SDK for code-writing agents, repeated the pattern with a narrower but more consequential blast radius. Its public `issue-analyze.yml` workflow read and summarized incoming issues using broad credentials, including an API key and Google Cloud service account key; an attacker who embedded injection text in an issue could cause that agent to post a `/adk-issue-fix` comment, which the privileged `issue-fix.yml` workflow treated as an authenticated command from a trusted collaborator [1][4]. The privileged job's command execution was restricted to a denylist permitting only `gh` and `git` invocations, an attempt to bound what a compromised agent could do even after triggering the workflow. That control failed in practice: `git`'s `-c core.hooksPath=<dir> hook run pre-commit` mechanism, along with `git -c "alias.x=!<command>" x`, allowed the agent to execute arbitrary shell commands using tooling the allowlist had explicitly approved, because the restriction validated the invoked binary rather than the full range of behavior that binary could produce [4]. The table below summarizes how the two generations of the attack differed while sharing the same root cause.

Dimension	Gemini-based workflows	Antigravity-based workflows
Public entry point	Pull request analyzed by triage agent	Issue analyzed by `issue-analyze.yml`
Trigger abused	`@gemini-cli` mention treated as maintainer command	`/adk-issue-fix` comment treated as maintainer command
Privileged identity	`adk-bot` – a human collaborator's personal access token	Job-scoped `GITHUB_TOKEN` plus dedicated agent credentials
Credentials at risk	Pull-request and issue write access	`ADK_TRIAGE_AGENT` PAT, `GITHUB_TOKEN`, `ADK_GCP_SA_KEY`
Execution control bypassed	Authorization based on comment origin, not content provenance	`gh` / `git` -only allowlist bypassed via `core.hooksPath` and `alias`
Primary impact	Forged review approvals, manipulated merge readiness	Credential exfiltration, arbitrary code execution on the CI runner

Both chains fail for the same underlying reason: the systems authorized action based on a *signal that looked like* a trusted instruction – a bot account name, a slash command, a recognized comment format – rather than verifying that the content producing that signal had not itself passed through an untrusted, attacker-controlled path. This is functionally identical to the "origin injection" pattern documented in other recent agent research, where an attacker forges authorship or provenance metadata so that a downstream system treats attacker content as if it originated from a trusted maintainer [9]. It is also a textbook instance of what security researcher Simon Willison termed the "lethal trifecta": the privileged Antigravity workflow simultaneously held access to sensitive credentials, was exposed to untrusted external content through the issue body, and had the ability to communicate its results externally by writing to the repository and its pull requests [10]. Removing any single leg of that triangle – narrower credential scope, stricter content isolation, or constrained output capability – would have substantially reduced the blast radius even if the prompt injection itself had succeeded.

Google's classification of the initial finding as non-rewardable for a financial payout — while still crediting the researcher with an Honorable Mention — on the grounds that exploitation required social engineering and that a maintainer would ultimately need to act on a poisoned pull request, is worth scrutinizing rather than accepting at face value. Traditional vulnerability triage assumes a human is the last line of defense against a socially engineered request; that assumption weakens considerably when the entity acting on the forged signal is itself an AI agent operating with standing credentials and no independent judgment about whether a request is unusual. The same reasoning that made Google comfortable calling this "social engineering" is precisely what makes agent-to-agent trust boundaries a distinct risk category deserving its own review criteria, separate from conventional single-actor exploitation.

Recommendations

Immediate Actions

Organizations operating AI agents inside CI/CD pipelines should audit every workflow trigger that can be satisfied by agent-generated content — comments, labels, commit messages, or status checks — and confirm that none of these signals alone are sufficient to invoke a more privileged workflow. Any bot account used to gate privileged automation should be verified as a genuine non-human identity rather than a human collaborator's personal access token, since the latter conflates a real contributor's authority with an automated process that adversarial content can influence. Command allowlists intended to constrain agent tool use should be reviewed against known escape techniques; a denylist or allowlist scoped to a binary name (`git` , `gh`) is not equivalent to a restriction on that binary's behavior, and mechanisms like custom hooks paths, aliases, and configuration overrides can turn an approved tool into an unrestricted shell.

Short-Term Mitigations

Credential scoping deserves particular attention: workflows that process untrusted external content, such as public issues and pull requests, should never hold long-lived cloud service account keys or broadly scoped tokens, and should instead operate with short-lived, narrowly permissioned credentials issued only for the specific action being taken. Where an agent-to-agent handoff must exist — one agent's output triggering another agent's privileged action — organizations should insert an out-of-band verification step, such as a cryptographically signed authorization token issued at the point a human or a genuinely trusted system initiates the request, rather than relying on pattern-matching a comment

format. Security teams should extend existing prompt injection test suites and red-team exercises to explicitly cover multi-agent chains, since defenses validated only against single-agent prompt injection will not catch a payload designed to propagate through an intermediate agent.

Strategic Considerations

At a structural level, this incident argues for treating agent-to-agent communication with the same skepticism applied to communication from an untrusted human actor, rather than granting implicit trust because a message originates from inside the automation pipeline. Provenance and data-flow tracking — verifying not just what an instruction says but where the content that produced it actually came from — should be treated as a maturing control category for any organization operating chained or multi-agent systems, not an optional hardening step reserved for the highest-risk deployments. Vendors and internal platform teams building agent orchestration frameworks should also reconsider how vulnerability disclosure programs classify agent-to-agent findings: dismissing a structural trust-boundary failure as "non-rewardable social engineering," even when paired with non-monetary recognition such as an Honorable Mention, risks under-incentivizing exactly the kind of research that surfaces these gaps before they are exploited at scale in production.

CSA Resource Alignment

This incident maps closely to a recent CSA threat intelligence publication on agent trust failures and to CSA's agentic threat modeling and controls guidance. CSA's Agent Data Injection: A New Attack Class Beyond Prompt Injection documents "origin injection," in which an attacker forges authorship or provenance metadata so that a downstream system or agent treats attacker-controlled content as though it came from a trusted maintainer [9]. That is precisely the mechanism behind the ADK case: a comment formatted to resemble a maintainer's `@gemini-cli` mention or `/adk-issue-fix` command was accepted as an authoritative trigger without verifying that the account or process producing it had not itself been manipulated. CSA's paper argues that agents hardened against direct prompt injection remain exposed when the attack targets the metadata layer instead of the instruction layer, a distinction the ADK incident illustrates directly.

CSA has also written specifically about applying its MAESTRO Agentic AI Threat Modeling Framework to CI/CD pipelines, providing the structural vocabulary for reasoning about failures that occur specifically at the boundary between agents rather than within a single agent's context window [11]. The ADK incident is a failure at MAESTRO's Agent Frameworks and Deployment and Infrastructure layers: the orchestration logic connecting a public triage agent to a privileged maintenance agent lacked a control point that

treated the inter-agent handoff itself as a trust boundary requiring verification. Organizations building or auditing multi-agent CI/CD pipelines should use MAESTRO's layered decomposition to explicitly model these hand-off points rather than assuming that per-agent hardening is sufficient once multiple agents are chained together.

Finally, CSA's AI Controls Matrix (AICM) v1.1 offers the governance layer that connects this incident to broader control ownership: its identity-and-access-management and supply-chain domains cover exactly the gaps exposed here, including non-human identity issuance for automation accounts, least-privilege scoping of service credentials, and third-party/tooling risk assessment for AI-agent-driven CI/CD dependencies. Mapping the immediate and short-term recommendations above to specific AICM control objectives gives security and audit teams a concrete, auditable path for closing the gap rather than treating the fix as a one-time patch to the ADK repository [12].

References

[1] The Hacker News. "Google Deletes 3 ADK AI Workflows After Malicious GitHub Issue Could Trigger Privileged Agent." The Hacker News, August 3, 2026.

[2] The Register. "Google dev kit spurs first-ever agent-on-agent violence." The Register, August 3, 2026.

[3] CSO Online. "Google ADK flaws reveal what happens when AI agents trust the wrong message." CSO Online, August 2026.

[4] Pillar Security. "I'll Just Call You: Agent-to-Agent Privilege Boundary Failures in CI/CD on Google's ADK Repository." Pillar Security Blog, August 3, 2026.

[5] Google. "google/adk-python: An open-source, code-first Python toolkit for building, evaluating, and deploying sophisticated AI agents." GitHub, 2026.

[6] Pillar Security. "My Agentic Trust Issues: From Prompt Injection to Supply-Chain Compromise on gemini-cli." Pillar Security Blog, May 5, 2026.

[7] GitHub Advisory Database. "Gemini CLI: Remote Code Execution via workspace trust and tool allowlisting bypasses (GHSA-wpqr-6v78-jr5g)." GitHub, April 24, 2026.

[8] Google. "Agent Development Kit (ADK) documentation." Google, 2026.

[9] Cloud Security Alliance. "Agent Data Injection: A New Attack Class Beyond Prompt Injection." Cloud Security Alliance, July 17, 2026.

[10] Willison, Simon. "The lethal trifecta for AI agents: private data, untrusted content, and external communication." Simon Willison's Weblog, June 16, 2025.

[11] Cloud Security Alliance. "Applying MAESTRO to Real-World Agentic AI Threat Models: From Framework to CI/CD Pipeline." Cloud Security Alliance, February 11, 2026.

[12] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

