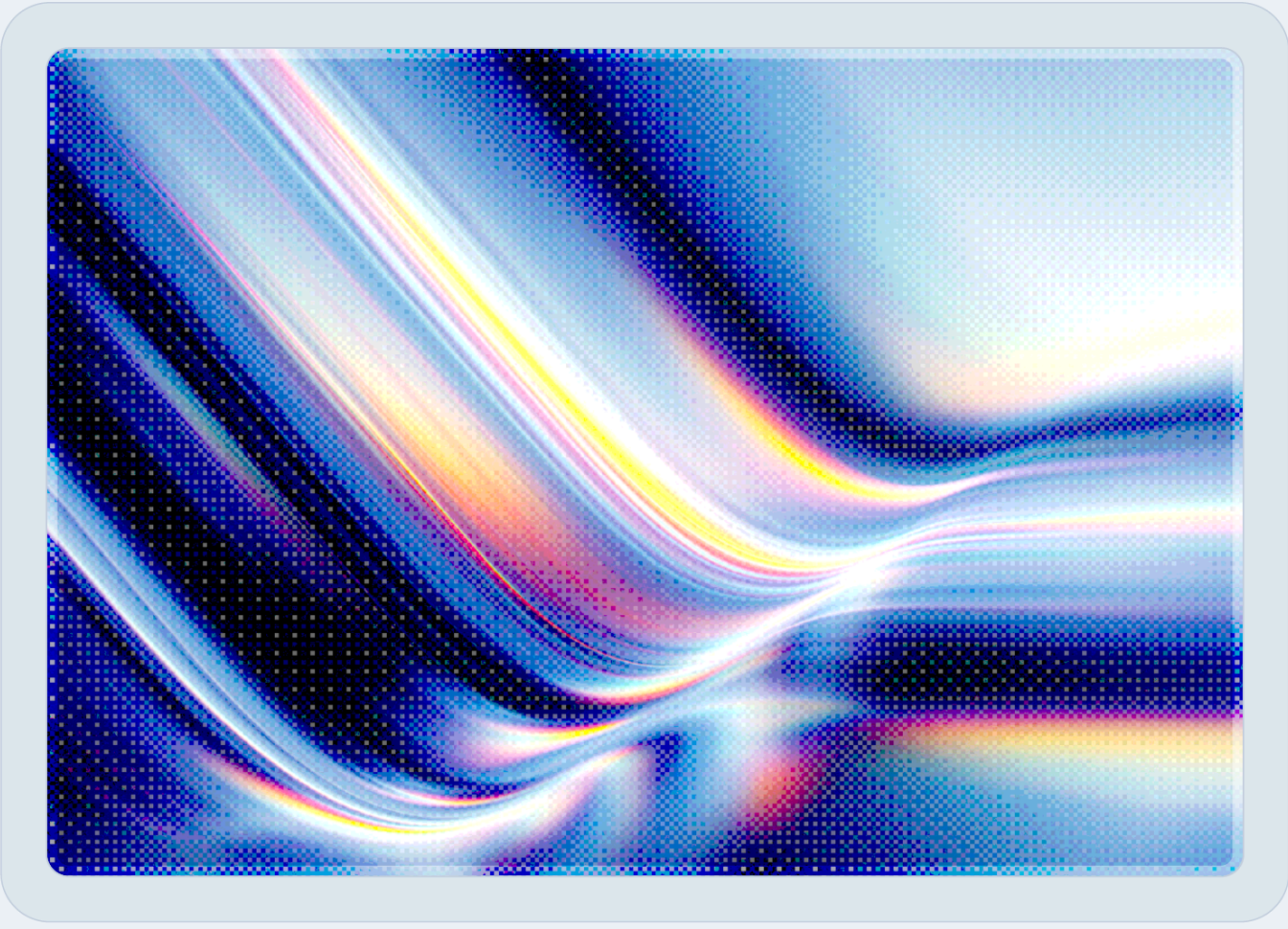
CSAI Foundation | Cloud Security Alliance

Rogue Agent: The Dialogflow CX Flaw That Hijacked AI Chatbots

A Shared Code-Execution Runtime Turned One Permission Into Cross-Agent Compromise

2026-07-14

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Security researchers at Varonis Threat Labs privately disclosed a critical vulnerability, dubbed "Rogue Agent," in Google Cloud's Dialogflow CX conversational AI platform in November 2025; Google shipped an initial fix in April 2026 and completed remediation in June 2026, with public disclosure following in July 2026 [1][2][3]. The flaw let anyone holding a single, seemingly modest permission on one Dialogflow agent overwrite a shared code-execution file and silently compromise every other Code Block-enabled agent running in the same Google Cloud project, regardless of which team or business unit owned those agents [2]. Once in place, the injected code could read live conversation transcripts, collect session variables containing customer data, and rewrite chatbot responses to solicit passwords or other credentials from users, all while leaving no trace in the customer's own Cloud Logging console [1][2]. Two additional weaknesses compounded the impact: the shared runtime had unrestricted outbound internet access that bypassed VPC Service Controls, and exposure of the Instance Metadata Service allowed retrieval of access tokens tied to a Google-managed service account [1][2]. No evidence of in-the-wild exploitation was found before the patch, and no CVE identifier was assigned to the issue [2][3]. The case is a concrete illustration of a pattern the CSA AI Safety Initiative has flagged repeatedly this year: agentic AI platforms often inherit multi-tenant isolation assumptions from earlier cloud services that do not hold once agents can execute arbitrary code on a shared host.

Background

Dialogflow CX is Google Cloud's platform for building widely used conversational agents, deployed across customer support, financial services, and healthcare contexts to power voice and text interactions at scale [2][3]. Within Dialogflow CX, "Playbooks" define the conversational logic an agent follows, and a feature called Code Blocks allows developers to embed custom Python logic directly inside a conversation flow, for example to call an external API, transform a variable, or apply business rules mid-conversation [2]. Google designed Code Blocks to let non-engineering teams extend chatbot behavior without waiting for a full application release cycle [2], a capability that also means the platform is executing customer-authored code as a routine part of serving a conversation.

Varonis Threat Labs researcher Daniel Reyhanian and colleagues examined how Google isolates that code execution across the many agents a single customer might operate within one Google Cloud project [2]. Their working assumption was that each agent's Code Block would run in a properly isolated

sandbox so that a change to one agent could not affect another. Investigation showed that assumption did not hold in Dialogflow CX: all Code Blocks belonging to agents in the same project shared a single Google-managed Cloud Run service as their execution environment [1][2]. That shared environment, rather than being a stateless, ephemeral sandbox, exposed a writable file system to the code running inside it, including a file named `code_execution_env.py` that Google's own backend relied on to actually run each Code Block's Python logic [2].

This combination — a shared runtime, file-system write access, and a persistent file that the platform itself trusted — set up the conditions for the vulnerability that followed. The researchers also found that the same Cloud Run service had unrestricted outbound internet access despite customers' VPC Service Controls perimeters, and that the Instance Metadata Service inside the environment was reachable — an exposure that IMDS access controls are specifically designed to prevent, since retrieving IMDS-based tokens can leak credentials tied to the service account behind them [1][2]. Both of these were treated as compounding findings alongside the core Rogue Agent flaw rather than separate, unrelated bugs.

Security Analysis

The attack chain required only one precondition: an attacker needed `dialogflow.playbooks.update` permission on any single Code Block-enabled agent within a Google Cloud project [1][2][3]. This permission is often granted broadly in practice — to developers, contractors, or administrators trusted to edit a chatbot's conversational logic, not to execute arbitrary code with reach into other teams' agents — which is what made the gap between apparent and actual scope consequential rather than merely theoretical. In our assessment, that gap between the apparent scope of a permission and its actual reach is the central lesson of this disclosure. In practice, a permission that looked like a content-edit right functioned as a code-execution right once that code ran inside a shared, writable environment [1][2].

With that access, an attacker could author a Code Block that, on execution, silently downloaded a modified version of `code_execution_env.py` from an attacker-controlled server and overwrote the legitimate file in the shared Cloud Run environment [1][2]. Because every Code Block across every agent in the project routed through that same file when the platform invoked Python's `exec()` function to run embedded logic, the attacker's tampered version began executing in the same trust scope as Google's own backend code for every subsequent conversation, across agents the attacker had never touched or had permission to edit [2]. The attacker could then restore the visible configuration of their own agent to appear unmodified, while the malicious logic persisted invisibly in the shared runtime [2]. Cloud Logging did not capture the file overwrite itself, which researchers described as making detection from the customer side "nearly impossible" absent the vendor's own telemetry [2].

Once embedded, the malicious code had two primary avenues of harm. First, it could inspect the variables Dialogflow CX populates during a conversation — full transcript history and session parameters — giving the attacker access to whatever a customer had typed or a bot had said, including personally identifiable information, payment details, or other sensitive fields depending on the deployment [1][2]. Second, by manipulating the `respond()` function that other agents' Playbooks call to generate bot output, the attacker could inject their own text into a live conversation, for example prompting a user mid-session to "re-enter your password" under the guise of the legitimate agent [1][2]. For organizations running Dialogflow CX in customer support, banking, or healthcare contexts, that combination effectively amounts to a silent, scalable phishing and data-exfiltration capability delivered through a channel users already trust.

The two compounding issues extended the potential blast radius beyond data visible inside the conversation itself. The shared execution environment's unrestricted outbound network access meant that even organizations relying on VPC Service Controls to constrain where their cloud workloads could send data found that control bypassed for anything running inside a Code Block [1][2]. Separately, the reachable Instance Metadata Service allowed retrieval of access tokens belonging to the Google-managed service account backing the Cloud Run environment, which — depending on that account's actual privileges — could extend an attacker's reach beyond the conversation data itself and into other Google Cloud resources the service account could touch [1][2]. Neither Varonis nor Google reported evidence that either weakness, or the core Rogue Agent flaw, was exploited by threat actors before the patch was completed [2][3].

Two structural factors make this disclosure instructive well beyond Dialogflow CX specifically. The first is that the vulnerability lived entirely inside Google's managed backend — based on the sources reviewed, customers had no way to detect, audit, or remediate it themselves through standard tooling, and no local configuration change could have closed the gap; only the vendor's own fix resolved it [1][2]. The second is that the flaw was not a defect in any individual agent's code but an emergent property of how multiple independently-owned agents were made to share a single execution substrate. CSA's AI Safety Initiative views this as illustrative of a broader class of risk we expect to recur: as vendors let customers embed custom logic into AI-driven workflows, isolation boundaries between tenants, agents, and conversation sessions face novel stress that traditional cloud multi-tenancy models were not designed for.

Recommendations

Immediate Actions

Organizations running Dialogflow CX with Code Block-enabled agents should confirm they are on Google's fully patched configuration as of the June 2026 resolution and should not assume the April 2026 interim update alone closed the gap [2][3]. Security teams should review Cloud Logging for `dialogflow.playbooks.update` actions across the relevant project's history, focusing on updates made by accounts outside the expected owning team for each agent, since Google and Varonis both recommended this as the primary forensic starting point for organizations wanting reassurance that the flaw was not exploited against them [2]. Any organization that granted playbook-edit permissions broadly — to contractors, cross-functional teams, or via overly permissive IAM roles — should treat this incident as cause to audit who actually holds `dialogflow.playbooks.update` today and whether that population is still appropriate.

Short-Term Mitigations

Beyond the immediate patch verification, organizations should reassess how they scope permissions across agents that share a Google Cloud project. Where feasible, segregating unrelated Dialogflow agents into separate projects reduces the population of agents any single compromised credential or over-privileged account could reach, even against isolation flaws not yet discovered. Teams operating Code Block logic should also review what data those blocks are permitted to access and log; since the underlying execution environment is vendor-managed and not directly observable, minimizing the sensitivity of data that ever flows through a Code Block's variables reduces the value of any future compromise of the same class. Where customer-facing conversations touch regulated data — health information, payment card data, or other categories subject to breach-notification obligations — organizations should confirm with their compliance function whether the exposure window (November 2025 disclosure to June 2026 full resolution) requires any notification review, even absent confirmed exploitation.

Strategic Considerations

Rogue Agent is a reminder that the security properties buyers assume from "the cloud" — strong tenant and workload isolation — cannot be taken for granted once a platform lets customer-supplied code execute inside a shared backend service, a pattern CSA's AI Safety Initiative believes is likely to recur as agentic AI tooling increasingly allows customer-supplied code execution, not just in Dialogflow CX. When

evaluating any platform that allows embedded scripting, custom tools, or code execution as part of an AI agent's behavior, security and procurement teams should ask vendors directly whether that execution is sandboxed per-agent, per-tenant, or shared, and what evidence the vendor can provide of that isolation being tested. Because this class of flaw is invisible to the customer by design, organizations should also push vendors for meaningful platform-side audit logging of any privileged action — including code and configuration changes deep in a managed runtime — rather than assuming an absence of visible logs means an absence of risk. The incident reinforces why CSA's guidance treats permission boundaries in agentic systems as a distinct, non-obvious risk category: a right that looks scoped to one narrow function, such as editing a chatbot's conversational script, can in practice be a code-execution right with reach far beyond its apparent purpose.

CSA Resource Alignment

This incident maps closely to threat categories the CSA AI Safety Initiative has already catalogued for agentic AI systems. The Agentic AI Red Teaming Guide organizes agent-specific vulnerabilities into twelve threat categories [4][8], two of which describe almost exactly what Rogue Agent demonstrated in production: "Agent Authorization and Control Hijacking," where a permission intended for one narrow function is exploited to gain broader control, and "Multi-Agent Exploitation," where trust relationships between co-located agents allow compromise to spread beyond the agent an attacker was originally authorized to touch [4][8]. Security teams testing their own agentic AI deployments should use that guide's methodologies for these two categories specifically, since the Dialogflow CX case shows both risks are not theoretical.

CSA's Data Security Within AI Environments guidance [5], which maps data protection controls to the AI Controls Matrix (AICM) across the full AI lifecycle, is directly relevant to the data-exposure half of this incident. The paper's proposed control extensions for prompt injection defense and its broader treatment of session isolation failures describe the same failure mode Rogue Agent exploited: conversation-scoped variables intended to stay within one session or one agent's boundary becoming visible to logic the customer never authorized to run. Organizations should use that paper's AICM control mapping to evaluate whether their own conversational AI deployments have equivalent session-isolation and data-minimization controls in place, independent of any single vendor's runtime architecture.

The permission-scoping failure at the root of this vulnerability — where `dialogflow.playbooks.update` functioned as a de facto code-execution grant rather than a content-edit grant — is also the central pattern documented in CSA's Identity and Access Gaps in the Age of Autonomous AI survey [7], which found that AI agents frequently borrow human or shared identities rather than being managed as distinct entities, allowing permissions to propagate

unexpectedly across agents. That same failure mode falls within the identity and access management domain of the AI Controls Matrix (AICM) v1.1 [6], CSA's current superset framework covering AI-specific security controls including the Cloud Controls Matrix. Organizations conducting vendor risk assessments of agentic AI platforms should use both resources together: the survey to benchmark whether their own identity governance practices are keeping pace with the autonomy they are granting agents, and AICM's IAM domain controls to explicitly ask whether a vendor's permission model separates configuration-editing rights from code-execution rights, since this incident shows those two can be conflated in ways that are not visible from a platform's documented permission names alone.

References

[1] The Hacker News. "Rogue Agent Flaw Could Have Let Attackers Hijack Google Dialogflow CX Chatbots." The Hacker News, July 2026.

[2] Reyhanian, Daniel, and Varonis Threat Labs. "Rogue Agent: How a Single Code Block Could Hijack Your AI Conversations in Google's DialogFlow." Varonis, July 2026.

[3] Dark Reading. "Dialogflow CX 'Rogue Agent' Flaw Enabled AI Chatbot Data Theft." Dark Reading, July 2026.

[4] Cloud Security Alliance. "Agentic AI Red Teaming Guide." Cloud Security Alliance, 2025.

[5] Cloud Security Alliance. "Data Security Within AI Environments." Cloud Security Alliance, 2025.

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

[7] Cloud Security Alliance. "Identity and Access Gaps in the Age of Autonomous AI." Cloud Security Alliance, 2026.

[8] Mello, John P., Jr. "Red-Teaming Agentic AI: New Guide Lays Out Key Concerns for AppSec." ReversingLabs Blog, June 2025.

