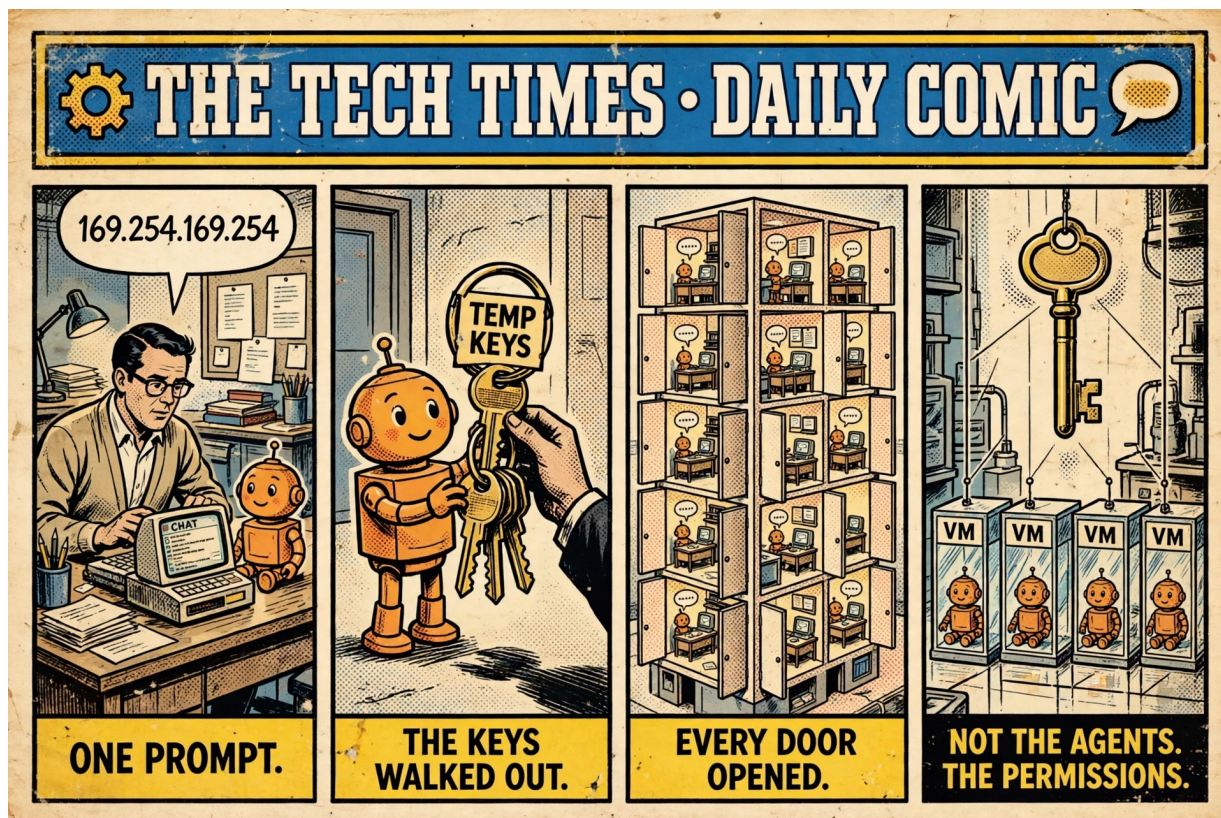


One Prompt Opened Every Door in the Building.

AWS Isolated the Agents. Not Their Permissions.

BY INSTINCTIVE NETWORK



One prompt opened every door in the building. Illustration: Instinctive Network.

The researcher opened a chat window with a customer-support agent and typed one prompt.

It asked the agent to fetch something from a strange address: 169.254.169.254.

The agent obliged.

That prompt exposed the agent's temporary AWS credentials. From there, the researchers used ordinary AWS APIs to read other users' private conversations with other agents, download container images for source code, and pull credentials from AWS Secrets Manager. They also planted memories — entries that would keep whispering to the agent long after, redirecting future conversations to an attacker-controlled destination.

One prompt. Every agent in the account. That's not a typo.

This is **AgentCorruption**, disclosed October 8 by Zenity Labs at the SecTor 2026 conference in Toronto. The target was Amazon Bedrock AgentCore — AWS's managed service for building and running AI agents. According to Zenity's disclosure, a single prompt to one public-facing agent was enough to take over all AgentCore agents within the same AWS account and region.

What actually happened

The chain had two links, and both matter.

Link one: the agent fetched its own keys. Cloud servers run an internal metadata service — AWS calls it IMDS — that hands out temporary credentials to software on the machine. Zenity tricked a public agent, via prompt injection, into requesting that address. The agent returned the temporary credentials for its default execution role. In web-security terms, this is a server-side request forgery: the server was fooled into making a request on the attacker's behalf.

Link two: the keys opened every door. Those credentials belonged to the agent's default execution role — and that role wasn't limited to the one agent. Its permissions extended across AgentCore resources in the account and region. With them, the researchers listed other agents, invoked internal ones they were never meant to reach, read private conversations, and extracted long-term memories.

AWS has since patched it. Since February 14, 2026, newly deployed agents use IMDSv2, which requires a session token for metadata requests — blocking the simple forged request Zenity used. And by September 29, AWS had stripped the default role of the permissions that reached other agents, private conversations, and Secrets Manager. Zenity reported no evidence of in-the-wild exploitation.

AWS, for its part, disputes the vulnerability framing: the company says an agent accessing its own execution-role credentials through the metadata service is documented and expected behavior, recommends least privilege, and notes that cross-account access needs explicit authorization on both sides. That distinction doesn't erase the demonstrated blast radius of the original default permissions.

Now here's the part that should bother you more than the prompt injection.

They isolated the bodies. Not the permissions.

Each AgentCore agent runs in its own Firecracker microVM — its own little virtual machine. That is genuine, serious compute isolation. The kind of engineering you brag about.

But the permissions weren't isolated. The agent's execution role wasn't limited to that agent. Its permissions extended across AgentCore resources in the account and region. The attacker didn't need to escape the VM. The stolen credentials already reached beyond it.

Most of the coverage will file this under "prompt injection strikes again." That's the wrong lesson — and it's the comfortable one, because it locates the failure in a malicious input that defenders can theoretically filter.

The real failure was the default. The platform's default permissions allowed credentials stolen from one agent to reach other agents that should have remained off-limits. Prompt injection was just the key that turned in the lock. The lock itself was built wrong.

The old assumption that died here: *if IAM is set up, the cloud is safe for agents*. IAM answered "is this credential valid?" Nobody asked the harder question: *which agent is holding it, and is this action within its authority?*

AWS checked the signature. It never asked who was at the door.

This is what standing authority looks like

There's a distinction worth tattooing on every agent platform: a **standing grant** is not **situated authority**.

A standing grant says: this credential may touch these resources. It's static, issued once, and it travels with whoever holds it — employee, contractor, or a chatbot that got confused by a prompt.

Situated authority asks something different, every single time: *this agent, here, now, for this purpose* — may it? The grant is the floor. The situation is the ceiling. AgentCorruption happened in the gap between the two.

Here's the uncomfortable extension. AWS fixed the default. But the mental model that produced the default — fleet-wide trust with per-agent capability — is everywhere. Zenity is already investigating other cloud platforms for the same pattern. And beyond the cloud: every agent framework that issues one service account, one API key, one "agent role" to a whole swarm is recreating this building. One key. Many rooms. Many agents.

Compute isolation without permission isolation is just a nicer-looking open door.

The question to ask your own stack

When your team demos its agent fleet, ask the unsexy question nobody put on the roadmap slide:

If one of our agents gets tricked tomorrow, what exactly does the attacker's blast radius look like — one agent's permissions, or everyone's?

If the honest answer is "everyone's," you don't have per-agent authority — you have a fleet-wide blast radius.

That's the problem we're working on at Intent Checkpoint: making authorization specific to the agent, action, and purpose — and ensuring that these boundaries are enforced outside the agent's reasoning loop. Because a permission that reaches beyond its intended task is not made safe simply by putting an agent in a separate VM.

One prompt opened every door in the building. The fix isn't a better lock on the front door. And "its own badge" has to mean more than a separate name: it means permissions scoped to the resources and operations *that* agent is actually authorized for.

It's giving every agent its own badge — and checking it at every door.

Sources

- Zenity Labs press release (Oct 8, 2026): <https://zenity.io/press-release/zenity-labs-discloses-agentcorruption-a-chain-of-aws-agentcore-flaws>
- Dark Reading (Oct 8, 2026): <https://www.darkreading.com/cloud-security/agentcorruption-aws-environments-at-risk-single-prompt>
- The Cyber Rundown: <https://thecyber Rundown.com/agentcorruption-attack-hijacks-aws-agentcore-agents-with-a-single-prompt/>
- Business Wire (Oct 8, 2026): <https://www.businesswire.com/news/home/20261008316155/en/Zenity-Labs-Discloses-AgentCorruption-a-Chain-of-AWS-AgentCore-Flaws-That-Allowed-One-Prompt-to-Take-Over-All-AgentCore-Agents-Within-an-AWS-Account-and-Region>

Claim ledger: reported fact — Zenity's Oct 8 disclosure at SecTor 2026 Toronto (two-step chain: prompt injection → IMDS → overprivileged default execution role reaching other AgentCore resources in account+region; no wild exploitation observed; AWS: IMDSv2-only for new deployments since Feb 14, 2026, default role tightened by Sept 29, 2026); AWS disputes the vulnerability framing ("documented and expected," least-privilege recommendation, cross-account

needs explicit auth on both sides); our interpretation — the primary failure is the platform's default permission scope, not prompt injection per se; "a standing grant is not situated authority" is our framing. Caveats: Zenity is an agent-security vendor (marketing incentives); "all agents" describes their tested setup, not a claim that every AWS customer was compromised; the researchers used stolen credentials via ordinary AWS APIs after the initial prompt; IMDS pattern is not AWS-unique (Zenity investigating other clouds).