CSAI Foundation

Cloud Security Alliance AI Safety Initiative

Antigravity Groundfall: Prompt Injection to RCE Chain

How Native Tool Injection Bypassed Google's Agentic IDE Security Boundary

Unofficial AI-assisted Research

2026-04-21

Key Takeaways

Pillar Security's April 20, 2026 disclosure documents a prompt injection to remote code execution (RCE) attack chain in Google Antigravity, Google's agentic IDE, in which insufficient input sanitization of the `find_by_name` tool's Pattern parameter allows an attacker to inject command-line flags into the underlying `fd` utility, converting a routine file search into arbitrary code execution [1]. The attack bypasses Antigravity's Secure Mode — the product's most restrictive security configuration — because the vulnerable tool is classified as a "native tool" rather than a shell command, placing its execution outside the boundary that Secure Mode evaluates [2]. We use the term "native tool injection" to describe this class of attack, in which an adversary exploits insufficient sanitization of parameters passed by an agent-native tool to an underlying OS utility, bypassing security controls that operate at the shell-command boundary. A separate disclosure by Mindgard, issued one day after Antigravity's November 2025 launch, documented a distinct persistent backdoor vulnerability in which malicious workspace rules can overwrite the global MCP configuration file, achieving code execution that survives project closure, application restart, and full reinstallation [3].

Taken together, these findings define a class of vulnerability that is likely to recur across agentic IDEs: trust boundaries drawn between "native" and "shell" tool invocations are architecturally incomplete when native tools reach the OS through parameters they do not sanitize. Organizations deploying agentic development environments should treat this as an active threat requiring architectural mitigations beyond security mode toggles:

- Apply available patches and monitor Google's security advisory channel for a CVE assignment for the `find_by_name` vulnerability.
- Audit every native tool parameter surface in deployed agentic IDEs for unsanitized flag-injection paths to underlying system utilities.
- Restrict agent access to global configuration directories and prohibit automatic execution of newly written configuration files on application launch.
- Treat repositories from untrusted sources as capable of containing indirect prompt injection payloads; sandbox or quarantine their contents before agent ingestion.

Background

Google Antigravity is an agentic integrated development environment announced on November 18, 2025, alongside the release of Google's Gemini 3 model family. Unlike traditional AI coding assistants, Antigravity is designed around an "agent-first" paradigm in which Gemini 3.1 Pro, Claude Sonnet 4.5, or GPT-OSS models operate as autonomous actors capable of planning, executing, and validating complex engineering tasks with minimal human intervention [4]. The platform offers two primary operational modes: an Editor view resembling conventional IDEs, and a Manager view for orchestrating multiple agents working asynchronously across parallel workspaces. Antigravity achieved 76.2% on SWE-bench Verified at launch, placing it among the top-ranked agentic development platforms on that benchmark at the time of release [5].

Antigravity's security model centers on a feature called Secure Mode, described as the product's most restrictive configuration. When Secure Mode is active, agent operations run through a virtual sandbox, network access is throttled, and code writes outside designated working directories are prohibited. Users and organizations could reasonably infer from the presence of a named, explicit security mode that activating it would contain the blast radius of agent operations and protect against adversarial manipulation of the agent's tool use. The Pillar Security disclosure demonstrated that this inference was incorrect for a specific class of tool invocation.

The Cisco State of AI Security 2026 report found that 83% of organizations plan to deploy agentic AI while only 29% feel prepared to do so securely — a deployment gap that expands the attack surface faster than defenses are being adopted [6]. Agentic IDEs represent a concentrated embodiment of this tension: they run with the full privileges of the developer's workstation, they process large volumes of external content as context, and their agent loops translate model reasoning directly into file system and process operations. The Antigravity vulnerabilities are among the first formally disclosed architectural vulnerabilities for a major agentic IDE — though neither has been assigned a CVE identifier as of this writing — and the attack patterns they document are generic to the architecture.

Security Analysis

The find_by_name Attack Chain

The Pillar Security vulnerability exploits a specific gap in how Antigravity's `find_by_name` native tool passes the user-supplied Pattern parameter to the underlying `fd` command-line utility. The `fd` utility is a modern, high-performance alternative to Unix `find`, and like most command-line tools it supports

a set of flags that modify execution behavior — including the `-X` (exec-batch) flag, which instructs `fd` to execute a specified binary against all matching files. Because Antigravity does not prepend the argument terminator `--` before the Pattern value and performs no flag-character filtering, any Pattern value beginning with `-` is interpreted by `fd` as a command-line flag rather than as a search string [1].

The three-stage attack chain proceeds as follows. First, an attacker stages a malicious payload file in the agent's workspace — for example, a script named `a.orange` containing an OS command. Second, the attacker causes the agent to invoke `find_by_name` with a crafted Pattern value such as `-Xsh`, which `fd` interprets as the exec-batch flag followed by a shell interpreter argument. Third, `fd` matches workspace files against the (now empty or minimal) search criteria and executes the staged payload against those matches, launching the attacker's code without any further user interaction [1]. The practical effect is arbitrary code execution with the same privileges as the running IDE process, which on developer workstations commonly means access to repository credentials, API keys, cloud provider tokens, SSH keys, and other sensitive materials that may be present in the development environment.

The Secure Mode bypass property distinguishes this vulnerability from conventional tool-parameter flaws. Secure Mode's restrictions operate at the shell-command invocation boundary: they evaluate commands before executing them and apply sandboxing constraints to commands that match the policy. The `find_by_name` tool, however, is classified internally as a "native tool" — a category of direct OS API invocations that bypass the shell command evaluation path entirely. When the agent calls `find_by_name`, Secure Mode's evaluation layer never receives the call. As Pillar Security researcher Dan Lisichkin stated in the disclosure: "The security boundary that Secure Mode enforces simply never sees this call" [1]. The result is that a user who deliberately chose the most restrictive available security configuration was nonetheless exposed to full RCE through a tool that the product's own architecture categorized as safe.

The indirect prompt injection vector extends this risk to scenarios involving no direct attacker access to the IDE's interface. An attacker who controls a public repository can embed adversarial instructions in source file comments — instructions that a developer's agent may execute when processing the file for context. Because agentic IDEs routinely ingest repository content to plan and execute coding tasks, any repository from an untrusted source becomes a potential injection delivery vehicle. The agent may execute the staged payload without any compromise of the developer's account or any explicit attacker interaction with the target environment [1].

Pillar Security reported the vulnerability to Google on January 7, 2026. Google acknowledged and accepted the report on January 24, deployed a patch on February 28, and awarded a bug bounty on March 26. The public disclosure occurred on April 20, 2026 [1]. At the time of this writing, Google has not published a CVE identifier for the vulnerability.

The Mindgard Persistent Backdoor

A second, architecturally distinct vulnerability was disclosed by Mindgard in November 2025, shortly after Antigravity's public launch. This vulnerability targets the `.agent` directory that Antigravity projects use to store user-defined agent rules — Markdown files whose contents are prepended to the system prompt with the explicit instruction that the AI "MUST ALWAYS FOLLOW WITHOUT ANY EXCEPTION." An attacker who can introduce a malicious file into a project's `.agent` directory can instruct the agent to copy a malicious MCP configuration file from the project into the global Antigravity configuration directory at `~/.gemini/antigravity/mcp_config.json`, registering a persistent MCP server entry that executes arbitrary system commands on every subsequent application launch [3].

The persistence mechanism is the defining characteristic of this vulnerability. Unlike a conventional prompt injection attack that executes a payload during the current session, the Mindgard technique survives project closure, Antigravity restart, and complete uninstallation and reinstallation of the application — because the malicious configuration file remains in the global directory even after the application is removed. Without a product-level patch, the primary user-level remediation is manual deletion of the corrupted configuration file [3], a step that most affected users would have no indication to take. Exploitation succeeds regardless of Antigravity's security settings: the vulnerability bypasses the Terminal Execution Policy "Off" setting, Review-Driven Development mode, Agent Non-Workspace File Access restrictions, and the Artifact Review Policy requiring human review before artifact acceptance [3].

Google's initial response to the Mindgard disclosure was to close the report as "Won't Fix (Intended Behavior)," citing an existing known issue entry for "Data exfiltration through indirect prompt injection." Mindgard disputed this classification on the grounds that the vulnerability involves global configuration file writes, achieves persistence through post-uninstall execution, and creates a backdoor mechanism rather than performing data exfiltration. Following Mindgard's follow-up communications, Google reopened the ticket and forwarded it to the product team for evaluation [3], but no CVE was assigned and no patch was issued as of the disclosure date.

The Shared Architectural Root Cause

The Pillar Security and Mindgard vulnerabilities target different mechanisms — tool parameter injection and workspace rule abuse, respectively — but share a common architectural root: Antigravity's trust model grants elevated execution authority to certain tool categories and workspace-level configurations without applying the same adversarial scrutiny that it applies to explicit shell commands. In both cases,

the attacker's payload reaches the OS through a path that the product's security architecture categorized as safe, bypassing the controls that would have detected or blocked an equivalent explicit shell invocation.

This pattern generalizes beyond Antigravity. CVE-2026-26268, affecting Cursor, documents a similar attack chain in which prompt injection causes an agent to write malicious git hooks to the `.git/config` file, achieving out-of-sandbox RCE when git operations trigger the hook [7]. The Trail of Bits analysis of prompt injection to RCE pathways in AI agents identifies the core vulnerability class as any scenario where model output influences OS-level execution through a path that lacks the sanitization applied to conventional input vectors [8]. Each of these disclosures is, in effect, the same finding instantiated in a different product: the security boundary between the model's reasoning and the OS execution surface is incompletely drawn when it relies on tool-category classifications rather than on universal input validation.

Recommendations

Immediate Actions

Security teams operating environments where developers use Google Antigravity should apply available patches and monitor Google's security advisories for a formal CVE assignment and additional patch guidance stemming from the Pillar Security disclosure. Organizations that have already applied the February 28 patch should verify the patch is in effect by checking the running Antigravity version and confirming it post-dates that release. Environments that have not applied the patch should treat the `find_by_name` tool as a confirmed RCE vector and consider disabling Antigravity access for developers working with untrusted repository content until a patch is confirmed in place.

For the Mindgard persistent backdoor, which remains unpatched, teams should audit all developer workstations running Antigravity for unexpected content in `~/.gemini/antigravity/mcp_config.json`. Any MCP server entries in that file that were not explicitly configured by the developer should be treated as potentially malicious and removed. Organizations should establish a monitoring policy for this file path and alert on writes to it that originate from agent processes rather than direct user actions.

Developers should avoid cloning or processing repositories from untrusted sources within Antigravity's agentic workspace context until the indirect prompt injection vector is remediated. When untrusted repository content is required, it should be reviewed in a conventional, non-agentic environment before being introduced to an active agentic workspace.

Short-Term Mitigations

Platform and security engineering teams should audit all native tool parameters in deployed agentic IDE configurations for the class of flag-injection vulnerability identified by Pillar Security. The structural requirement is that every tool parameter that is ultimately interpolated into a command invocation must be preceded by the `--` argument terminator and must have `-`-prefixed strings filtered before reaching the underlying utility. This is a standard input sanitization control in CLI tooling, and its absence from Antigravity's `find_by_name` implementation is an instructive failure mode to check for across all comparable tools.

Agent access to global application configuration directories — particularly paths like `~/.gemini/antigravity/mcp_config.json` and equivalent locations in other agentic IDEs — should be restricted through filesystem-level access controls. On macOS and Linux, tools with agent process isolation capabilities can be configured to prevent agent subprocesses from writing to the user home directory outside designated workspace paths. On Windows, analogous restrictions can be applied through integrity levels or Software Restriction Policies. These controls do not require product-level patches and can be implemented unilaterally by security teams.

Organizations should extend their software composition analysis and supply chain security policies to cover agentic workspace content, in addition to conventional software dependencies. Repositories ingested by agentic IDEs should be treated with the same adversarial scrutiny applied to third-party packages, including review for embedded instructions in comments, README files, and documentation that could serve as indirect prompt injection vectors.

Strategic Considerations

The Antigravity disclosures surface a governance gap in how security mode classifications are communicated to users and security teams. When a product names and exposes a "Secure Mode" or equivalent high-assurance configuration, users and organizations reasonably infer that activating that mode provides meaningful protection against the most obvious attack classes. The Pillar Security finding demonstrates that this inference was wrong for Antigravity, and the Mindgard finding demonstrates it was wrong even for the basic persistence threat model. CSA recommends that AI IDE vendors publish explicit threat model documentation for each security configuration they expose, specifying which attack classes the mode does and does not address, and that procurement processes for agentic development tools include evaluation of this documentation.

The broader implication for the AI security community is that architectural trust boundaries in agentic systems require formal, adversarial specification rather than emergent categorization. The distinction between "native tools" and "shell commands" that underlies the Antigravity Secure Mode bypass is an

internal implementation detail that was never hardened against adversarial use. As agentic IDEs mature, vendors should adopt threat modeling frameworks — such as MAESTRO — that treat every tool's parameter surface as a potential adversarial input, regardless of the tool's internal classification, and should design execution boundaries accordingly.

CSA Resource Alignment

The CSA MAESTRO framework for agentic AI threat modeling directly addresses the vulnerability class documented in both Antigravity disclosures. MAESTRO's seven-layer threat model identifies the Agent Execution Environment (Layer 2) and the Tool Invocation Layer as high-severity surfaces requiring strict input validation and privilege scoping. The `find_by_name` flag injection attack is a concrete instantiation of MAESTRO's "tool parameter manipulation" threat: an adversary who cannot directly invoke a shell command achieves equivalent capability by manipulating the parameters of a trusted native tool. Had MAESTRO's recommended controls been applied — treating all tool parameter inputs as untrusted, applying argument terminators and flag-character filtering at tool boundaries, and mapping tool capabilities to minimum required privileges — the Pillar Security vulnerability class would have been mitigated [9].

MAESTRO also classifies prompt injection as a foundational threat to agentic integrity across multiple framework layers, emphasizing that the risk extends beyond LLM alignment into OS-level compromise wherever model output influences process execution. The indirect prompt injection vector in the Antigravity disclosure — where attacker-controlled repository content issues instructions to the agent — is precisely the attack class MAESTRO was designed to surface. Organizations applying MAESTRO to agentic IDE deployments should explicitly include indirect prompt injection via repository content and `.agent` directory rules in their threat model, and should test agent behavior against crafted adversarial inputs in those channels [9].

The CSA AI Controls Matrix (AICM) addresses the governance and operational dimensions through its Secure Development and Agent Execution Control domains. The Mindgard persistent backdoor — in which a malicious workspace rule achieves persistent RCE by writing to a global configuration directory — represents a failure of the AICM control objective for agent write access scoping: agents should not be permitted to modify global application state outside their designated workspace context without explicit human authorization. AICM control objectives for supply chain provenance and third-party content validation apply equally to repository content ingested by agentic workspaces, a category that most existing supply chain security programs have not historically treated as in-scope [10].

The CSA Zero Trust guidance is applicable to the broader pattern of implicit trust that both vulnerabilities exploit. Antigravity's native tool invocations receive elevated trust based on their category rather than on explicit per-invocation authorization. Zero Trust architecture requires that every request for privileged action — including tool invocations that reach the OS — be authenticated and authorized against a current policy rather than against a static category assignment. Implementing Zero Trust principles at the tool invocation layer would require the agent execution environment to verify, at invocation time, that the specific parameters being passed are consistent with the expected behavior for that tool, rejecting the call if parameters carry the structural characteristics of flag injection [11].

For organizations engaged in STAR-based AI vendor assessment, the Antigravity disclosures provide concrete evaluation criteria for agentic IDE security maturity: whether vendors maintain published threat models for security configurations, whether security modes are tested against the attack classes they purport to prevent, and whether vendors respond to architectural vulnerabilities with product-level controls or with developer responsibility frameworks. The Mindgard disclosure's "Won't Fix (Intended Behavior)" initial classification is a notable data point for vendor response quality assessment in agentic tooling procurement.

References

[1] Pillar Security. "Prompt Injection leads to RCE and Sandbox Escape in Antigravity." Pillar Security Blog, April 20, 2026.

[2] CyberScoop. "Vuln in Google's Antigravity AI agent manager could escape sandbox, give attackers remote code execution." CyberScoop, April 2026.

[3] Mindgard. "Forced Descent: Google Antigravity Persistent Code Execution Vulnerability." Mindgard Blog, November 2025; updated April 2026.

[4] Google Developers. "Build with Google Antigravity, our new agentic development platform." Google Developers Blog, November 2025.

[5] index.dev. "Google Antigravity: The Agentic IDE Changing Development Work." index.dev, 2026.

[6] Cisco. "Cisco State of AI Security 2026." Cisco Blog, 2026.

[7] NIST National Vulnerability Database. "CVE-2026-26268 Detail." NVD, 2026.

[8] Trail of Bits. "Prompt injection to RCE in AI agents." Trail of Bits Blog, October 2025.

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

[10] Cloud Security Alliance. "AI Controls Matrix." CSA, 2025.

[11] Cloud Security Alliance. "Zero Trust Guidance for Critical Infrastructure." CSA, 2024.