CSAI Foundation

Cloud Security Alliance AI Safety Initiative

Antigravity Sandbox Escape: Prompt Injection and Native Tool Abuse

How Native Tool Trust Gaps Enabled Remote Code Execution in Google's Agentic IDE

Unofficial AI-assisted Research

2026-04-22

Key Takeaways

- Researchers at Pillar Security discovered a prompt injection vulnerability in Google's Antigravity agentic IDE that allowed arbitrary code execution by injecting the `-X` (exec-batch) flag into the `find_by_name` tool's Pattern parameter — converting a file-search operation into an arbitrary binary execution call [1][2].
- The exploit bypassed Antigravity's most restrictive security mode, Strict Mode, because `find_by_name` is classified as a native system tool and fires before any command-level security evaluation occurs, illustrating how pre-evaluation trust grants to built-in tools create a privileged but unsandboxed execution path [1][3].
- A complete attack chain requires no elevated access: an attacker need only cause Antigravity to ingest a malicious document, repository file, or web page containing adversarial instructions, after which the agent orchestrates file creation and executes the injected payload autonomously [2].
- A parallel vulnerability, "Forced Descent," disclosed by Mindgard, achieved persistent code execution through a separate mechanism — exploiting global configuration files that Antigravity executes at startup regardless of project trust state, surviving even full application reinstallation [4].
- Both vulnerabilities were responsibly disclosed and patched by February 28, 2026, and recognized through Google's Vulnerability Reward Program, but the attack classes they represent — native tool privilege bypass and startup configuration abuse — are not specific to Antigravity and are likely to recur across the growing ecosystem of agentic developer tools, as the parallel Cursor findings illustrate [1][4][7][8].

Background

Google announced Antigravity on November 18, 2025 [4], alongside the release of the Gemini 3 model family, positioning it as an agent-first development environment [5]. Unlike earlier AI-assisted coding tools that generated suggestions for human approval, Antigravity delegates multi-step software development tasks to autonomous agents that operate across three command surfaces simultaneously: the code editor, an integrated terminal, and a built-in browser [5][6]. The platform is a heavily modified

fork of Visual Studio Code, supports Gemini 3.1 Pro and Flash models as well as Anthropic Claude Sonnet 4.5 and OpenAI's GPT-OSS, and was made available in public preview at no cost to individual developers [5]. A Manager view enables orchestrating multiple agents working in parallel across workspaces, supporting asynchronous multi-agent task execution — a capability not present in prior-generation IDE AI plugins [6].

Antigravity's capability profile makes it a qualitatively different security boundary than a conventional code editor. When an Antigravity agent operates on a developer's machine, it holds the authority to read and write arbitrary workspace files, execute shell commands, interact with browser windows, and invoke a suite of native tools that expose filesystem operations, process execution, and network access. Google designed Strict Mode — the platform's most restrictive security configuration — to constrain this surface by limiting network access, blocking out-of-workspace writes, and ensuring all command operations execute within a sandboxed context [1]. Strict Mode was positioned as the appropriate operational posture for agents processing untrusted content, such as open-source repositories or third-party web resources. The Pillar Security disclosure demonstrated that this posture offered less protection than its design intended.

The agentic IDE market had attracted significant security research attention by the time the Antigravity vulnerabilities were disclosed, as evidenced by the Cursor CVE series documented in 2025 [7][8]. Cursor, a competing AI-augmented editor, accumulated its own series of significant findings during 2025 and early 2026, including CVE-2025-59944 — a case-sensitivity bypass that allowed attackers to overwrite protected configuration files on Windows and macOS by exploiting the gap between case-insensitive filesystem semantics and case-sensitive security policy enforcement — and the CurXecute and MCPoison family (CVE-2025-54135, CVE-2025-54136), documented by Check Point Research and Tenable, which exploited Model Context Protocol configuration injection for arbitrary code execution [7] [8]. The Antigravity disclosures are therefore best understood not as isolated incidents but as evidence of a pattern: as agentic tools acquire more capability and deeper system integration, the attack surface they expose appears to be expanding faster than the security models governing them [9], a pattern consistent with the broader history of capability-security gaps in rapidly adopted developer tooling.

Security Analysis

The Pillar Security Discovery: Flag Injection Through a Native Tool

The exploit documented by Pillar Security exploits how Antigravity's `find_by_name` tool passes developer-supplied input to the underlying `fd` filesystem search utility [1][3]. The tool accepts a Pattern parameter, which the implementation interpolates directly into the `fd` command line without sanitizing strings that begin with the `-` character. Because UNIX-style utilities treat arguments beginning with `-` as flags rather than literal values, an attacker controlling the Pattern parameter can inject arbitrary `fd` flags rather than supplying a search pattern.

The critical injected flag is `-X`, which instructs `fd` to execute a specified binary against each file it matches in the workspace. By supplying a Pattern value of `-Xsh`, an attacker causes `fd` to pass each matched workspace file to the `sh` shell interpreter for execution. The attack is constrained by one structural limitation: spaces within the injected Pattern string are treated as literal characters rather than argument separators, preventing an attacker from passing arguments directly to the `-X` flag within the Pattern field alone [1]. This constraint does not prevent exploitation; it merely changes the composition of the attack chain. Because Antigravity permits file creation as an unrestricted native action, an attacker can first direct the agent to write a shell script to the workspace, then invoke `find_by_name` with a pattern that matches that file and executes it [1][2].

The attack's most security-relevant characteristic is its relationship to Strict Mode, Antigravity's own highest-assurance protection layer. That security configuration is intended to prevent exactly this class of unauthorized code execution. However, `find_by_name` is classified internally as a native system tool, a category that executes before Antigravity's command-level security controls evaluate the operation [1][3]. The Strict Mode runtime never has the opportunity to inspect or block the `find_by_name` invocation because, by the time any protective layer evaluates command semantics, the native tool has already fired. This is not a subtle configuration error but a structural gap: the trust model appears to grant built-in tools pre-evaluation execution authority — implicitly assuming native tools are inherently safe — without sanitizing the parameters through which those tools accept user-controlled input [1][3].

Indirect Prompt Injection as the Initial Vector

Pillar Security's disclosure characterizes the vulnerability as reachable through prompt injection, meaning that an attacker need not have direct access to a developer's Antigravity session to initiate the attack chain [1][2]. Antigravity, like other LLM-powered agents, does not reliably distinguish between data it is processing for context — the content of a file being read, a web page being summarized, a code repository being reviewed — and instructions it should execute [9]. An adversarial instruction embedded in an open-source repository README, a malicious web page, a crafted code comment, or a document shared via a compromised identity connector can direct Antigravity to invoke `find_by_name` with a malicious Pattern value without any action by the legitimate developer beyond initiating a task that causes the agent to ingest that content [2].

This delivery mechanism is significant because it lowers the barrier to exploitation to near zero for attackers capable of placing content in repositories or websites that developers are likely to process with Antigravity. Supply chain compromise of popular open-source packages — already a documented threat vector independently of AI tooling — gains an additional dimension when agentic IDEs read package source code and documentation as part of routine development workflows. Any task that causes Antigravity to ingest third-party content — summarizing a library's API, generating tests against a dependency — can serve as the delivery mechanism if that content has been seeded with adversarial instructions [2].

Forced Descent: A Distinct and Persistent Attack Path

Mindgard disclosed a separate vulnerability class in Antigravity under the name "Forced Descent" that operates through a different mechanism and produces a more durable compromise [4]. Where the Pillar Security finding requires prompt injection to trigger tool misuse, Forced Descent exploits the global configuration architecture of Antigravity itself. The platform permits projects to define custom rules in configuration files — a legitimate feature that allows development teams to codify project-specific agent behavior. Antigravity reads and applies these rules at startup for every user message, prior to any project trust verification or workspace isolation check [4].

An attacker who distributes a repository containing malicious configuration rules can achieve code execution against any developer who opens that repository in Antigravity, because the rules execute before the user has been given any opportunity to evaluate or approve the project's contents. More significantly, Mindgard found that Antigravity's global configuration directory — from which configuration files are loaded on every application launch — can be written to via the same file-creation capability that Pillar Security identified as a component of the flag injection chain [4]. Once a malicious `mcp_config.json` has been written to the global configuration directory, it executes on every

subsequent Antigravity startup, regardless of whether the user opens a new project. Mindgard confirmed in controlled testing that such a configuration file persists through a full application uninstall and reinstall cycle, because the global configuration directory is not removed as part of the application's uninstall procedure [4]. Remediation requires a developer to be aware of the compromise and manually delete the configuration file — a standard assumption of system integrity that the vulnerability directly undermines.

The two vulnerabilities together constitute overlapping attack paths with a shared enabling condition: the agent's ability to write files to locations that influence its own subsequent behavior. This capability boundary — where an agent's file-write authority extends to paths that affect the agent's own configuration or execution — represents a self-modification risk that existing security models — including traditional filesystem access control frameworks — do not explicitly address, as they were not designed with autonomous file-writing agents in mind.

The Broader Agentic IDE Threat Landscape

The pattern demonstrated by the Antigravity findings is not vendor-specific. Academic research synthesizing 78 studies on prompt injection attacks against agentic coding assistants (2021–2026) reports attack success rates against state-of-the-art defenses exceeding 85% when adaptive attack strategies are applied [9]. Separate research specifically addressing agentic coding editors documents vulnerability to prompt injection via GitHub pull request content, including PR titles, issue bodies, and issue comments, affecting GitHub Copilot Agent, Anthropic's Claude Code Security Review feature, and Google's run-gemini-cli, with exploitation enabling API key and access token theft [9]. Cursor's CVE-2025-59944, patched in version 1.7, illustrates that even non-AI-specific logic errors — a case-sensitivity mismatch between the security policy layer and the filesystem — can produce code execution when embedded in an agentic tool context, because the agent acts on behalf of the user with the user's full filesystem permissions [7].

The unifying characteristic across these findings is that agentic developer tools have been brought to market with capability profiles — autonomous file write, shell execution, browser control, identity-connected API access — that substantially exceed the capability profiles of the prior generation of IDE plugins, while the security architectures revealed by these disclosures appear to have adapted existing approaches — sandboxing, permission gating, content filtering — rather than developing models that account for the unique properties of autonomous agents, as the Antigravity Strict Mode bypass illustrates. Current research suggests prompt injection is unlikely to be resolved as a perimeter control problem in these environments: it is a consequence of LLMs' inability to reliably distinguish instructions from data, a property that independent researchers have documented across architectures and that appears unlikely to change on a timescale commensurate with the current deployment pace of agentic tools [9].

Product	Vulnerability	Mechanism	Outcome	Status
Google Antigravity	Pillar Security (Jan 2026)	Prompt injection → `find_by_name` flag injection	Sandbox escape, RCE	Patched Feb 28, 2026 [1]
Google Antigravity	Forced Descent (Mindgard)	Global config file write via agent file-creation	Persistent RCE at startup	Patched [4]
Cursor	CVE-2025-59944	Case-sensitivity bypass of config file protections	RCE via config overwrite	Patched in v1.7 [7]
Cursor	CVE-2025-54135/54136 (CurXecute, MCPoison)	MCP configuration injection	Arbitrary code execution	Patched [8]
Copilot Agent, Claude Code, run-gemini-cli	PR content injection (2025–2026)	Prompt injection via issue/PR text	API key/token theft	Ongoing class [9]

Recommendations

Immediate Actions

Organizations and individual developers currently using Google Antigravity, Cursor, or comparable agentic IDEs should confirm they are running patched versions. Antigravity users should verify their installation reflects the February 28, 2026, patch for the Pillar Security vulnerability; the Mindgard Forced Descent finding was addressed in the same release cycle [1][4]. Developers should inspect the Antigravity global configuration directory for unexpected `mcp_config.json` entries or custom rules files not introduced by their own teams. On macOS and Linux, this directory resides at

`~/.gemini/antigravity/`; on Windows, under the `.gemini\antigravity\` path within the user profile directory [4]. Any configuration files whose provenance cannot be traced to a known, trusted source should be treated as potentially malicious and removed before restarting the application.

Developers who have recently used Antigravity to analyze untrusted repositories, process open-source package source code, or summarize third-party web content in the period before the February 28 patch should treat their development environment as potentially compromised and conduct a review of recently created files in workspace directories and in the global configuration path.

Short-Term Mitigations

Security and engineering teams deploying agentic IDEs at enterprise scale should enforce Strict Mode as a policy baseline while recognizing that Strict Mode alone does not eliminate the attack surface revealed by this disclosure. Until vendors demonstrate that native tool input sanitization is comprehensive and independently verified, organizations should treat agentic IDE environments as partially trusted even under the most restrictive available mode.

Where possible, run agentic IDE sessions in isolated environments — dedicated virtual machines, containerized developer workstations, or cloud-hosted development environments — that separate the agent's file-write authority from paths affecting the agent's own startup configuration and from filesystems containing production credentials, signing keys, or deployment tokens. This structural separation is more reliable than relying on the agent's internal policy enforcement, which the Antigravity findings demonstrate can be bypassed at the native tool layer. Inventory and restrict the identity connectors and API tokens that agentic IDE instances hold; the blast radius of a successful prompt injection or tool abuse is directly bounded by the access those connections provide.

For repositories processed by agentic IDEs — whether internal codebases or third-party dependencies — apply the same content-trust verification that is applied to software supply chain artifacts more broadly. README files, documentation, code comments, and CI configuration files are all surfaces through which adversarial instructions can be delivered to an agent that reads repository content as part of task execution. Organizations should review their open-source dependency consumption practices in light of the indirect prompt injection vector documented by Pillar Security.

Strategic Considerations

The Antigravity vulnerabilities illustrate a structural problem unlikely to be resolved by any single vendor patch cycle: agentic tools derive their value from the same properties — autonomous action, broad system access, the ability to process and act on arbitrary content — that make them difficult to secure

against adversarial misuse. Security teams should engage with their engineering organizations on a deliberate posture for agentic developer tool adoption, treating these tools as a new privileged software class rather than as extensions of existing IDE security policy.

Vendor security transparency should be a formal evaluation criterion when selecting agentic development platforms. The Antigravity disclosure is notable for a relatively responsible timeline — Pillar Security submitted the report on January 7, 2026, Google acknowledged it the same day, accepted it on January 24, issued a fix on February 28, and awarded a bug bounty on March 26 [1]. Organizations should prefer vendors who demonstrate comparable responsiveness, publish detailed security advisories, and operate active vulnerability reward programs for their agentic products.

Longer-term, the industry needs formal security models that address the unique properties of autonomous agents: the right to write files that influence agent behavior should be governed by explicit policy rather than inherited from general filesystem permission grants; native tool inputs should be sanitized to the same standard as inputs processed by security-evaluated components; and startup configuration loading should enforce provenance verification on configuration files. CSA's MAESTRO framework provides conceptual language for these requirements and should inform procurement questions and product security evaluations as the agentic IDE market matures.

CSA Resource Alignment

The Antigravity sandbox escape can be analyzed through multiple layers of the CSA MAESTRO framework, the primary threat modeling architecture CSA has developed for agentic AI systems [10]. The Pillar Security attack chain traverses at least three MAESTRO layers in sequence: the adversarial prompt instruction enters through Layer 1 (Foundation Model inputs and input surfaces), manipulates agent planning and tool selection at Layer 2 (Agent Frameworks and Planning), and achieves unauthorized execution at Layer 3 (Tool and API Use). The Forced Descent vulnerability adds a persistence dimension through Layer 4 (Memory and State), where the malicious configuration file constitutes a corrupted persistent state that survives session boundaries. MAESTRO treats cross-layer attack chains — those initiating at Layer 1 and cascading to Layer 3 and beyond — as a high-severity threat class, given the compounding of trust violations across agent subsystems [10]. The Antigravity findings are a textbook instantiation of this pattern in a production system.

The CSA AI Controls Matrix (AICM) addresses the control gaps this incident exposes across several of its 243 control objectives spanning 18 domains [11]. The most directly relevant domain is Model Security, which includes controls addressing prompt injection defense — a control category the AICM v1.0 release specifically identified as requiring expanded coverage given its emergence as a critical AI-specific risk

[11]. The AICM's Supply Chain Security domain addresses the indirect prompt injection vector: open-source repositories and third-party packages are supply chain artifacts, and the AICM's controls for third-party AI component provenance apply to the content those components carry as well as to the code itself. Organizations implementing AICM controls should extend their supply chain security scope to include documentation and metadata ingested by agentic tools, not only the software packages themselves.

CSA's AI Organizational Responsibilities guidance is relevant to the enterprise deployment questions this incident raises. The guidance addresses governance frameworks for AI tool adoption and the shared responsibility models applicable when commercial AI tools operate with organizational credentials and data access. The Antigravity case illustrates why governance frameworks must account for the agent's effective permission set — including identity connectors and API tokens — as a distinct risk dimension separate from the model's intrinsic capabilities [12].

References

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

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

[3] The Hacker News. "Google Patches Antigravity IDE Flaw Enabling Prompt Injection Code Execution." The Hacker News, April 21, 2026.

[4] Mindgard. "Forced Descent: Google Antigravity Persistent Code Execution Vulnerability." Mindgard Blog, 2026.

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

[6] AI for Developers. "Google Antigravity: The Agent-First IDE That Wants to Replace Your Entire Workflow." AI for Developers, 2025.

[7] Lakera. "Cursor Vulnerability (CVE-2025-59944): How a Case-Sensitivity Bug Exposed the Risks of Agentic Developer Tools." Lakera Blog, 2025.

[8] Tenable. "FAQ: CVE-2025-54135, CVE-2025-54136 Vulnerabilities in Cursor – CurXecute and MCPoison." Tenable Blog, 2025.

[9] arxiv.org. "Prompt Injection Attacks on Agentic Coding Assistants: A Systematic Analysis of Vulnerabilities in Skills, Tools, and Protocol Ecosystems." arXiv:2601.17548, January 2026.

[10] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 6, 2025.

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

[12] Cloud Security Alliance. "AI Organizational Responsibilities: AI Tools and Applications." Cloud Security Alliance, 2025.