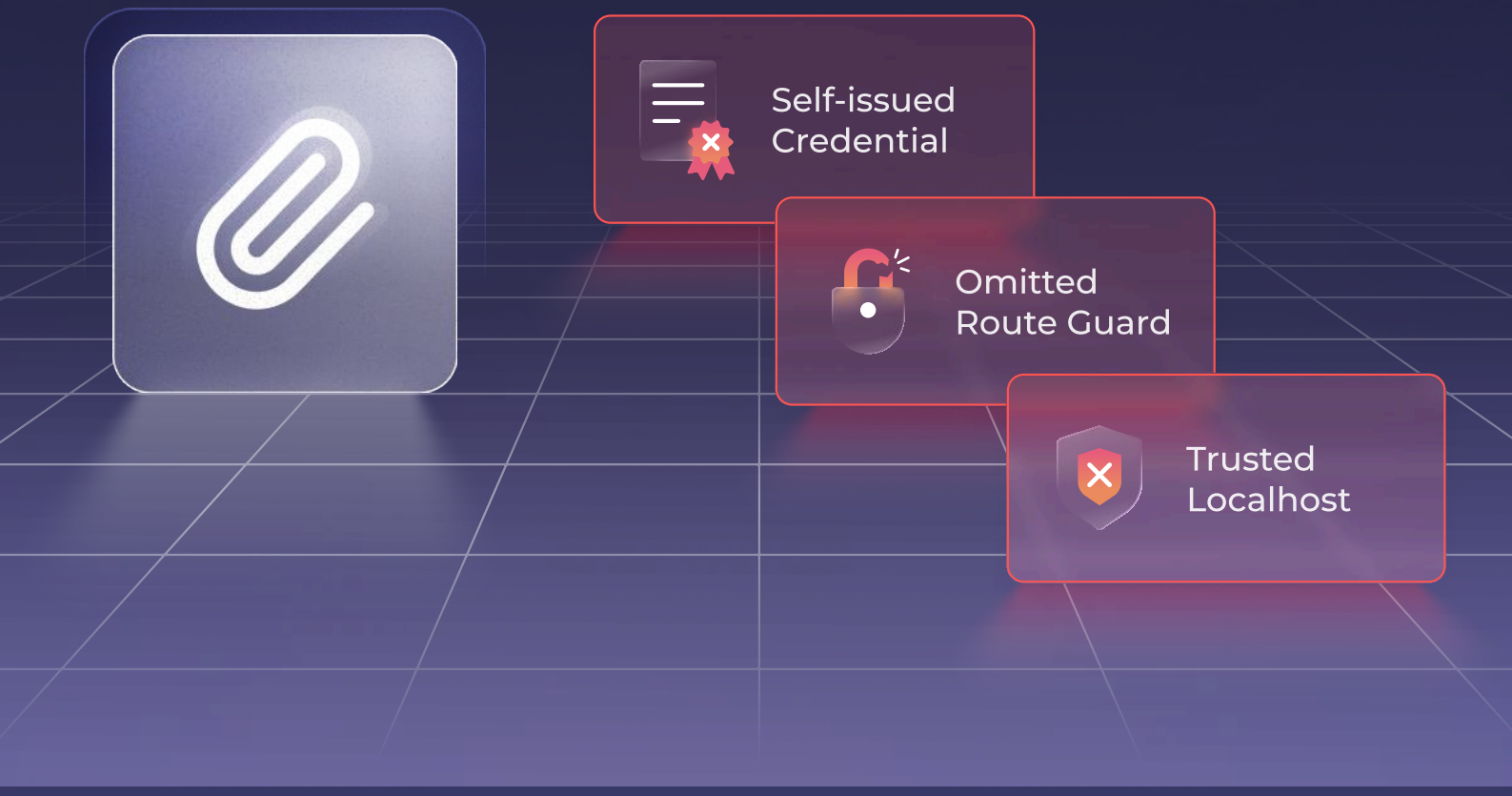
OASIS

Breaking the Paperclip

Critical Vulnerabilities in AI Agent Orchestration

A technical analysis of authorization failures, ungated APIs, and DNS rebinding leading to remote code execution in Paperclip AI.

Sagi Layani, Solutions Architect, Oasis Security · July 2026



OASIS

Contents

PART 01

Introduction

AI-agent platforms do more than generate text. They coordinate autonomous workers, connect them to enterprise systems, and often allow them to run commands on behalf of their operators. Agent platforms blur the boundary between configuration and execution. In Paperclip, agent configuration can determine which commands run on the server, so unauthorized control of that configuration can create a path to host-level code execution.

Paperclip is an open-source control plane for operating "zero-human companies." It organizes AI agents into companies, assigns work, and launches agents through configurable adapters. During a security assessment of Paperclip, we identified three vulnerabilities spanning its authenticated and `local_trusted` deployment modes. In authenticated mode, one attack chain turned self-registration into server-side command execution, while missing guards on several API endpoints exposed data and workflows. In `local_trusted` mode, DNS rebinding allowed an attacker-controlled webpage to cross the loopback boundary and execute commands on the developer's machine.

Although the three findings affected different surfaces, each relied on an implicit trust assumption: that a self-approved credential conveyed sufficient authority, that every sensitive route would enforce its own access checks, or that requests reaching a loopback service originated from trusted local software. In an agent control plane, failures at these boundaries can expose data and privileged workflows or provide a path to host-level code execution. This paper explains the attack chains, the product assumptions that enabled them, and the changes Paperclip introduced after disclosure.

Findings at a Glance

Finding	Trust-boundary failure	Result
Unauthenticated RCE via Authorization Bypass	An attacker could create an account, obtain elevated access, and introduce executable configuration through a privileged operation.	Remote command execution
Unauthenticated Access to API Endpoints	Sensitive routes omitted authentication or company-scoping checks.	Data and workflow exposure
DNS Rebinding RCE	An attacker-controlled webpage could reach a locally running instance and have its requests treated as administrator actions.	Command execution on the developer's machine

PART 02

Paperclip in Brief

Only four Paperclip concepts are needed to understand the findings.

Deployment modes. In authenticated mode, users sign in, and requests are authorized according to their identity, company membership, and instance-level privileges. In `local_trusted` mode, the default local-development experience, Paperclip assumes requests reaching the loopback service belong to the local operator and assigns them an implicit instance-administrator identity.

Companies and agents. A Paperclip instance can host multiple companies. Each company contains agents, tasks, and supporting configuration. The company is therefore both a tenant boundary and a container for executable agent behavior.

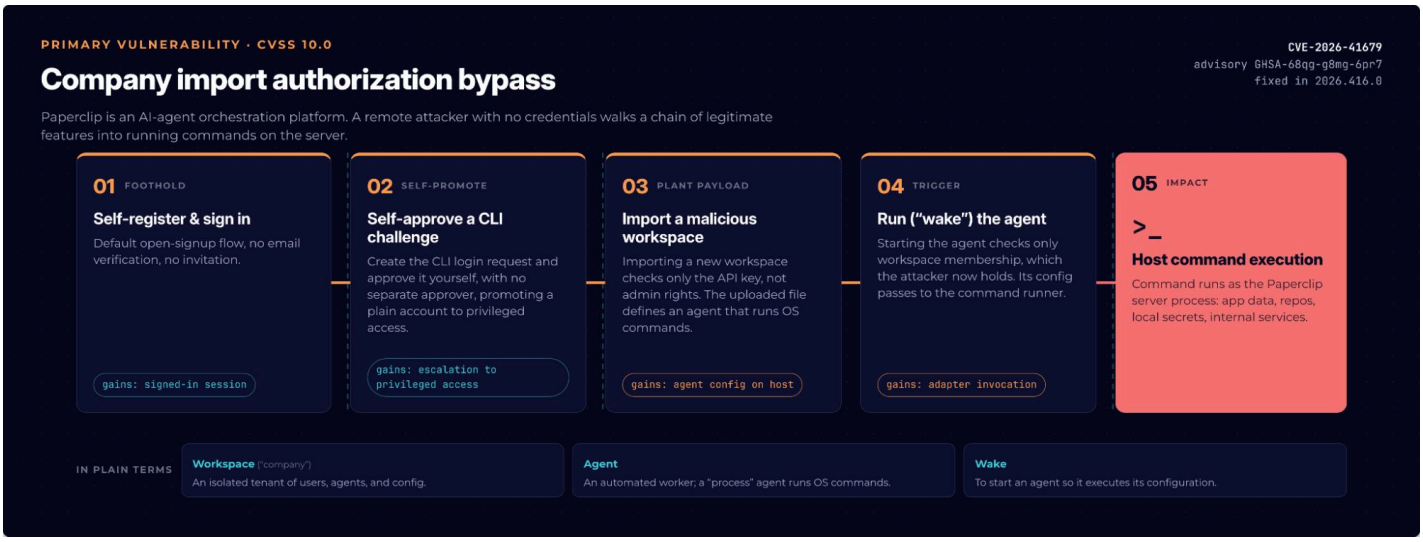
Company portability. Companies can be imported from portable bundles. A bundle may contain a `.paperclip.yaml` manifest that defines agents and their adapter configurations. Importing a company is, consequently, not just a data-upload operation; it can create runnable agents.

Adapters. Adapters connect Paperclip agents to their execution environment. The built-in `process` adapter intentionally launches a configured command as a child process of the Paperclip server. That capability is not itself a vulnerability; it is a legitimate execution sink that must only be reachable through correctly authorized configuration and activation paths.

Together, these concepts create the core security invariant: whoever can introduce and wake an agent with a local execution adapter can run code with the operating-system privileges of the Paperclip server.

PART 03

CVE-2026-41679: RCE via Company Import Authorization Bypass



> **Severity:** Critical — CVSS 10.0
>
> **Affected configuration:** Network-accessible authenticated deployments using the default registration configuration
>
> **Affected versions:** Earlier than 2026.416.0
>
> **Published advisory:** GHSA-68qg-g8mg-6pr7

CVE-2026-41679 was the most consequential finding because it connected a missing authorization check directly to Paperclip's intended agent-execution workflow. Under the default authenticated-mode configuration, a remote attacker could start without credentials and finish with arbitrary command execution as the Paperclip server's operating system user.

3.1 From Open Registration to a Self-Approved Board Key

Authenticated mode distinguished signed-in users from instance administrators. A newly registered user should not have been able to create top-level tenants or deploy privileged configuration, and a durable board API credential should have required an independent authorization decision.

Paperclip nevertheless allowed self-registration by default and did not require email verification. A network attacker could create an account and sign in immediately, without an invitation or control of a verified mailbox. That session was enough to enter Paperclip's CLI authorization flow, a challenge-and-approval process used to authorize a command-line client and activate a persistent board API credential.

The attacker could create a CLI authorization challenge without authentication. After signing in, the same user could approve that challenge themselves; Paperclip did not require a separate approver. Approval activated the pending board API token and tied it to the attacker's account, effectively converting a self-registered account into persistent board-level API access. That self-promotion was the bridge from open signup to the import endpoint.

3.2 The Missing Import Boundary

With a self-approved board API key in hand, the attacker could reach the company import routes. Paperclip correctly restricted direct creation of a new company to an instance administrator, but the equivalent new-company import path enforced only board-level access. It did not require instance-admin privileges.

That discrepancy mattered because an imported company could contain complete agent definitions. An attacker-controlled `.paperclip.yaml` could specify an agent using the `process` adapter together with a command and arguments chosen by the attacker. Paperclip accepted the bundle, created the company, created the configured agent, and granted the importing user access to the resulting company.

The import operation therefore collapsed several privileged actions into one request:

1. Create a new tenant-level company.
2. Define an agent inside that company.
3. Select a host-level execution adapter.
4. Supply the command that the adapter would launch.

Each action was part of a legitimate product workflow. The chain became critical because open signup and self-approved CLI authorization provided a remote attacker with the required board credential, while the import route treated that credential as sufficient for an operation equivalent to creating an instance-admin company.

3.3 Turning Configuration into Execution

After import, the attacker was a member of the newly created company. Paperclip's wakeup endpoint required company access, so the authorization check succeeded. Waking the imported agent passed its attacker-controlled configuration to the `process` adapter, which launched the specified command as the Paperclip server `process`.

The resulting attack chain was short:

1. Self-register and sign in through the default open-signup flow.
2. Create a CLI authorization challenge and self-approve it to activate a persistent board API key.
3. Use that key to import a new company containing a malicious process-based agent.
4. Use the returned identifier to wake the imported agent.
5. Paperclip executes the configured command on the host.

Successful exploitation provided the permissions of the service account running Paperclip. Depending on deployment, that could include access to application data, source repositories, local credentials, secrets exposed to agent processes, and internal services reachable from the host.

This chain illustrates why agent configuration must be treated as executable input. A portability bundle may look like declarative YAML, but fields that choose an adapter and its command are functionally equivalent to a program waiting to be invoked.

3.4 Root Cause and Fix

The root cause was an authorization mismatch between two semantically equivalent operations. Directly creating a company required instance-administrator access; importing a new company did not. Route-level authorization treated import as moving data rather than installing executable behavior.

Paperclip fixed the issue by applying a shared target-access check to both import preview and import execution. Imports into a new company now require instance-administrator privileges, while imports into an existing company require access to that company. The patch also tightened company scoping on related approval, activity, and heartbeat operations and added regression tests for the affected routes.

The fix shipped in version 2026.416.0. Open registration remained available for Paperclip's onboarding model, but a self-registered board user could no longer use the new-company import path as an instance-administrator equivalent.

PART 04

GHSA-xfqj-r5qw-8g4j: Ungated API Endpoints

> **Severity:** High — CVSS 8.3
> **Affected configuration:** `authenticated` mode
> **Patched version:** 2026.416.0
> **Published advisory:** GHSA-xfqj-r5qw-8g4j

The second advisory covered several routes that did not enforce the authentication or company-access rules expected in authenticated mode. Their individual impact varied, but together they exposed how Paperclip's route-by-route authorization model could fail open through omission.

Heartbeat run issues — `GET /api/heartbeat-runs/:runId/issues`. Each heartbeat run was associated with a company. A caller who knew a valid run identifier could retrieve the run's issue data without demonstrating authorization to access that company. This created a cross-tenant information-disclosure risk rather than unrestricted enumeration, because exploitation still required obtaining or discovering a valid run identifier.

Skill documentation — `/api/skills/*`. Unauthenticated callers could retrieve Paperclip's agent-facing instructions, including API paths, authentication conventions, and coordination procedures. This reduced the reconnaissance required to interact with an instance programmatically.

Health information — `GET /api/health`. The response disclosed deployment mode, exposure, version, authentication readiness, bootstrap state, and feature flags. These fields helped a remote attacker choose an appropriate strategy and identify potentially affected versions.

The underlying problem was structural. In authenticated mode, middleware could assign an unauthenticated request a "no actor" identity and continue processing it. Each route then had to remember the appropriate assertion. A missed assertion created an open endpoint.

The hardening added company-access validation to heartbeat-issue retrieval, authentication to general skill endpoints, invite-scoped alternatives for onboarding, and a reduced health response for unauthenticated users. It also expanded authorization regression coverage across sensitive routes.

PART 05

GHSA-x8hx-rhr2-9rf7: Drive-by RCE via DNS Rebinding

> **Severity:** Critical — CVSS 9.6
>
> **Affected configuration:** Default `local_trusted` mode
>
> **Affected versions:** Earlier than 0.3.1
>
> **Published advisory:** GHSA-x8hx-rhr2-9rf7
>
> **User interaction:** Required — opening an attacker-controlled webpage

The third vulnerability reached the same execution sink from the opposite deployment model. Paperclip's local mode bound its service to loopback and treated every request as an implicit instance administrator. That model assumed that a request reaching `127.0.0.1` must have originated from trusted local software. DNS rebinding allowed a remote website to violate that assumption.

5.1 Crossing the Loopback Boundary

In the demonstrated attack, the attacker-controlled hostname resolved to both the attacker's server and `127.0.0.1`. The browser initially connected to the attacker's server and loaded the exploit JavaScript. Once that server became unavailable, subsequent requests to the same hostname were retried against the loopback address. The browser still considered those requests same-origin, while Paperclip received the attacker-controlled hostname in the `Host` header.

Paperclip's server accepted arbitrary `Host` headers in `local_trusted` mode. Once a rebound request reached the local service, the authentication middleware assigned it the same implicit administrator identity as a legitimate local request. No cookie, API token, or stolen credential was necessary.

5.2 From a Webpage to Host Execution

The attacker served the initial page on Paperclip's default port, keeping the browser's origin stable after rebinding. Once the hostname resolved to the loopback address, the page could communicate with the local Paperclip API. It imported a company containing a `process`-based agent and then invoked the agent's wakeup endpoint. Because local mode treated both operations as administrator actions, Paperclip launched the configured command on the developer's machine.

The full path was:

1. The victim visited the attacker-controlled site.
2. After the attacker's server became unavailable, the browser retried the hostname against `127.0.0.1.`
3. The browser sent same-origin requests to the local Paperclip service using the attacker's Host header.
4. Paperclip implicitly authenticated those requests as an instance administrator.
5. The page imported and woke a malicious process-based agent.
6. Paperclip executed the attacker's command with the developer's privileges.

5.3 Root Cause and Fix

Three product behaviors formed the chain: local requests received administrator privileges, the server did not validate the hostname in this mode, and authorized agents could intentionally launch local processes. The execution adapter again amplified a boundary failure; it was not an accidental command-injection gadget.

Paperclip addressed the direct boundary failure by enabling hostname validation in `local_trusted` mode. A rebound request carrying an attacker-controlled Host header is rejected before it reaches the API. As additional defense in depth, Paperclip hardened imports by validating adapter types, preventing powerful `process` and `http` adapters in agent-safe imports, normalizing imported configuration, and placing newly imported agents into the company's approval workflow when required.

PART 06

One Pattern, Three Trust-Boundary Failures

The findings affected different routes and deployment modes, but all three depended on an implicit trust decision:

“A self-issued board credential is safe enough.” Open signup and self-approved CLI authorization allow an unauthenticated attacker to manufacture persistent board access; the import path then treats that credential as sufficient to create a tenant and install an executable agent configuration.

“Every sensitive route will remember its guard.” Authentication was opt-in at the handler level, so a single missing assertion exposed data or functionality.

“Localhost means the local user.” Loopback binding was treated as proof of identity even though a browser could relay remote content to a local service.

In each case, the assumption worked during the expected workflow. It failed when an attacker entered through a less obvious path: portability rather than direct creation, a forgotten route assertion rather than a sign-in flow, or a browser-mediated request rather than a local client.

The fixes reflect the controls that agent platforms need at these boundaries: equivalent operations must share an authorization policy; authenticated surfaces should default to denial except for a small, explicitly public set; local administrative services must validate Host and origin assumptions; and imported executable configuration should be validated and staged for approval.

Paperclip’s response combined narrow fixes with broader defense-in-depth. Version 2026.416.0 addressed the two authenticated-mode findings by enforcing administrator access for new-company imports and repairing confirmed company-scoping gaps. Subsequent hardening protected the local hostname boundary and constrained risky import behavior. Operators should upgrade beyond the affected versions and explicitly configure registration according to the intended exposure of their deployment.

PART 07

Managing AI Agent Access at Scale

AI agents are becoming a new class of enterprise identity. They act on behalf of employees, developers, and services; obtain or inherit credentials; choose tools; and make changes across code, cloud, SaaS, and internal systems. Their value comes from delegated authority. The security challenge is ensuring that authority remains attributable, appropriately scoped, and governed from human intent to agent action.

Traditional identity and access management usually evaluates a person or workload at a login or API boundary. Agentic workflows are more dynamic. A user may delegate a goal to an agent, the agent may invoke additional agents or tools, and each step may select a new credential or resource. By the time an action reaches the target system, it may see only a valid token, not the originating user, the responsible agent, the intended task, or the decisions made along the way.

Managing this model at scale requires four capabilities:

Discover agent identities and access paths. Security teams need a continuously updated view of the agents operating across the organization, who owns them, which human and machine identities they represent, what credentials they can use, and which resources those credentials can reach. Without this inventory, agent adoption creates a growing layer of shadow access.

Map delegation and ownership. Every action should remain connected to the human or service that initiated it, the agent that interpreted the request, and any downstream agent or tool it invoked. This relationship graph enables governing the entire chain rather than evaluating each credential in isolation.

Enforce contextual least privilege. Authorization should reflect the agent, initiating principal, requested task, target resource, risk, and time window, not merely whether a credential is valid. Agents should receive the minimum access required for the current objective, with stronger controls or approval when an action exceeds expected intent.

Preserve end-to-end attribution. Audit records should capture the request, agent identity, policy decision, credential used, tool invoked, and resulting change. Traditional endpoint or application logs often record only the final process or API call. They cannot explain why the action occurred or whether it was authorized in context.

These capabilities define Agentic Access Management: extending identity governance from static users and service accounts to autonomous systems that interpret intent and act across multiple resources. At organizational scale, the problem cannot be solved by patching each agent tool independently. Security teams need a consistent identity and policy layer that follows delegated authority wherever the agent operates.

Oasis Security's Agentic Access Management approach is designed for this layer. It provides visibility into agent identities and access paths, maps relationships from users and services through agents to enterprise resources, evaluates access using intent-aware context, applies least-privilege controls, and preserves a contextual audit trail.

OASIS

About Oasis Security

Oasis Security is the identity security platform for the AI era.

As enterprises adopt AI at scale, they face a new security challenge: thousands of machine identities and autonomous agents operating at machine speed, without the SSO, MFA, and governance controls that protect human access.

Oasis delivers unified discovery, policy intelligence, and lifecycle enforcement across hybrid environments, giving security teams the visibility to find what legacy tools miss, the context to understand what actually matters, and the automation to govern at the speed of AI. Backed by Accel, Cyberstarts, and Sequoia Capital, Oasis Security was founded in 2022 by Danny Brickman and Amit Zimerman.

The Oasis Research team

Our dedicated research team is committed to enhancing security in the field of identity. We take pride in our responsible and professional collaboration with vendors to address vulnerabilities and strengthen overall security.

The author

Sagi Layani, Solutions Architect at Oasis Security