OASIS

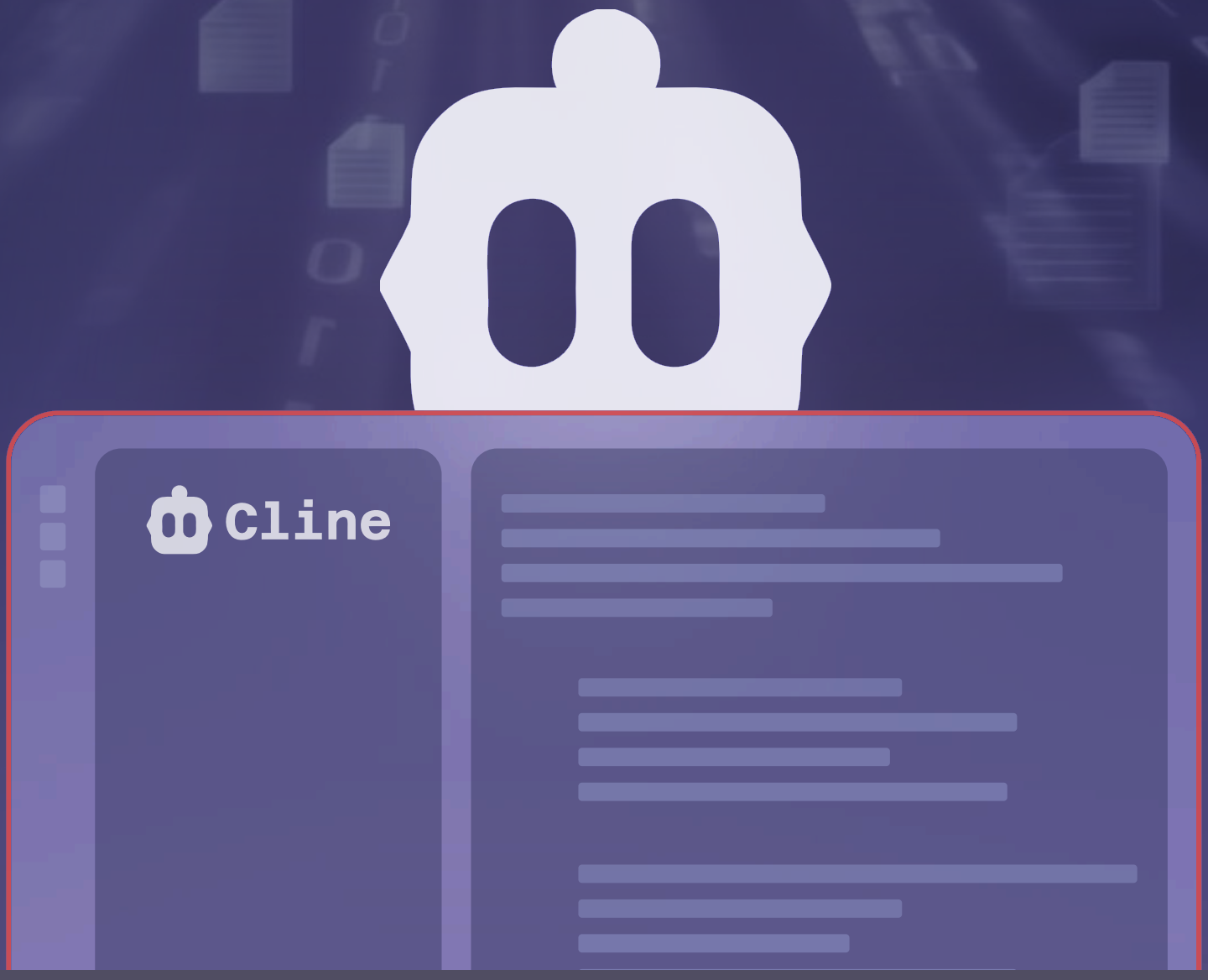
Hijacking the Agent

Cross-Origin WebSocket Exploitation in Cline Kanban

A Technical Analysis of Unauthenticated WebSocket Endpoints Enabling Information Disclosure, Remote Code Execution, and Denial of Service

Sagi Layani
Solutions Architect

Published on
May 7, 2026



OASIS

Overview

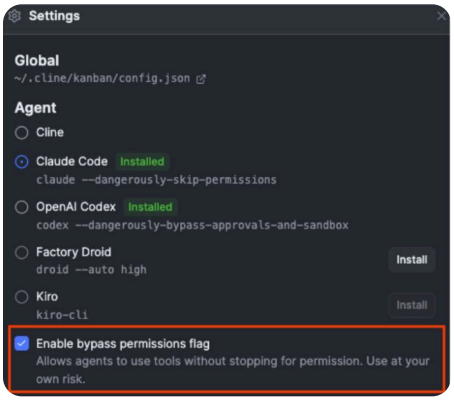
Introduction

AI coding assistants have become standard fixtures in the modern developer workflow. They have deep access to local codebases, execute shell commands, manage Git repositories, and interact with cloud services on behalf of their users. Their utility is proportional to the breadth of access they require, and that access creates a high-value target for attackers.

Cline is one of the most widely adopted open-source AI coding assistants. Its kanban feature introduces a web-based project management interface that orchestrates AI agent tasks through a local WebSocket server.

During a security assessment of the Cline kanban server, we identified a critical cross-origin WebSocket hijack vulnerability (CVSS 9.7) affecting all three WebSocket endpoints. Any website a developer visits while Cline is running can silently connect to the kanban server, exfiltrate workspace data, inject commands into the AI agent's terminal, or terminate active agent sessions.

By default, Cline is configured with a "bypass permissions" flag. This architecture allows the AI agent to execute shell commands and modify the filesystem without requiring manual, per-action user authorization. While this configuration facilitates the autonomous execution, it simultaneously eliminates a vital human-in-the-loop security control that would mitigate unauthorized command injection. This risk can be addressed by disabling the "Enable bypass permissions flag" within the application Settings.



This whitepaper provides a technical analysis of the vulnerability, explains the browser security model that makes it exploitable, walks through the complete attack chain, and discusses implications for organizations deploying AI coding agents at scale.

CVSS Score	**9.7 (Critical)**
CWEs	CWE-306 (Missing Authentication for Critical Function), CWE-1385 (Missing Origin Validation in WebSockets)
Affected package	`kanban` npm package v0.1.59
Repository	https://github.com/cline/kanban

Cline Kanban Architecture Overview

Understanding the vulnerability requires familiarity with the Cline kanban server's structure and how it communicates with the browser-based UI.

What Cline Kanban Does

Cline is an AI-powered coding assistant that operates from the command line. Its kanban feature extends the CLI with a web-based project management interface that starts a local HTTP and WebSocket server on `127.0.0.1:3484`. The web UI provides a board view for managing AI agent tasks: creating tasks, assigning them to the agent, monitoring execution, and reviewing results.

The kanban server is the coordination layer between the developer's browser and the AI agent's execution environment. It tracks workspace state, manages agent terminal sessions, and streams real-time updates to the UI.

WebSocket Endpoints

The kanban server exposes three WebSocket endpoints, each serving a distinct function:

Runtime State Stream (`/api/runtime/ws`): This endpoint provides real-time workspace state to the kanban UI. On connection, it immediately sends a full snapshot of the developer's environment, including project filesystem paths, workspace state (tasks, git information, board configuration), workspace metadata (git summary), and the cline session context version. After the initial snapshot, it streams live updates as the developer works: task state changes, AI agent chat messages, git activity, and board modifications.

Terminal I/O (`/api/terminal/io`): This endpoint provides raw bidirectional byte-stream access to the AI agent's pseudo-terminal (PTY). The kanban UI uses it to display agent output and to relay user input to the agent. Messages sent on this WebSocket are written directly to the agent's terminal input buffer.

Terminal Control (`/api/terminal/control`): This endpoint provides session management commands for agent tasks. It accepts structured JSON messages.

Authentication Model

There is none. The kanban server performs no authentication or authorization on any of its three WebSocket endpoints. It does not validate the Origin header on WebSocket upgrade requests. It does not require session tokens, API keys, or any form of credentials. Any TCP connection to port 3484 that sends a valid WebSocket upgrade handshake is accepted and granted full access to the endpoint's functionality.

The implicit assumption is that binding to `127.0.0.1` restricts access to the local machine, and that any connection originating from the local machine is the legitimate kanban UI. As we will demonstrate, this assumption is incorrect.

Background: WebSockets, Same-Origin Policy, and Browser Security

The exploitability of the Cline kanban vulnerability depends on how browsers handle WebSocket connections, particularly in relation to the Same-Origin Policy and CORS. This section provides the necessary background for readers who may not be familiar with these mechanisms.

The Same-Origin Policy and CORS

The Same-Origin Policy (SOP) is the foundational security mechanism in web browsers. It prevents a script loaded from one origin (defined as the combination of scheme, host, and port) from reading responses to requests made to a different origin. When JavaScript on `https://malicious-site.com` attempts to call fetch(`"http://127.0.0.1:3484/api/data"`), the browser blocks the script from reading the response.

Cross-Origin Resource Sharing (CORS) is the mechanism by which servers opt into cross-origin access. When a browser detects a cross-origin request, it sends a preflight OPTIONS request. The server responds with headers (`Access-Control-Allow-Origin`, etc.) indicating which origins are permitted. If the appropriate headers are absent, the browser blocks the response. Critically, CORS applies only to HTTP requests made via `fetch()` and `XMLHttpRequest`.

Why WebSockets Bypass CORS

WebSocket connections are established through an HTTP upgrade handshake, but once established, they operate outside the HTTP request/response model. CORS works by intercepting HTTP responses and evaluating their headers before allowing the calling script to read them. The WebSocket upgrade is a one-shot handshake that transitions the connection to a persistent bidirectional channel. There is no subsequent HTTP response to intercept.

When JavaScript calls `new WebSocket("ws://127.0.0.1:3484/api/runtime/ws")` from a page loaded on `https://malicious-site.com`:

1. The browser sends an HTTP GET with `Upgrade: websocket` to `127.0.0.1:3484`. The browser includes the `Origin` header (set to `https://malicious-site.com`) but does not perform a CORS preflight.
2. If the server responds with `101 Switching Protocols`, the WebSocket connection is established. The browser does not check for `Access-Control-Allow-Origin` headers.
3. The calling script has full read/write access to the WebSocket channel.

The **critical implication: the browser sends the Origin header in the upgrade request, but the server is responsible for validating it**. If the server does not check the Origin header and reject connections from unauthorized origins, any webpage can establish a WebSocket connection. The browser provides the information; the server must act on it.

The WebSocket RFC (RFC 6455, Section 10.2) explicitly states that servers "SHOULD" validate the Origin header to prevent cross-origin attacks. In practice, omitting Origin validation on a localhost WebSocket server is a critical vulnerability.

The Localhost WebSocket Attack Surface

The combination of these factors creates a consistent attack surface: any developer tool that exposes a WebSocket server on localhost without Origin validation is vulnerable to cross-origin hijacking from any website the developer visits. The browser provides no automatic protection. The defense must come from the server.

This is not a theoretical concern. The OpenClaw vulnerability demonstrated the same attack pattern against another AI agent platform's localhost WebSocket server. The Cline kanban vulnerability is another instance of an established class that persists because the WebSocket security model is widely misunderstood.

Vulnerability Analysis

The Cline kanban server exposes three WebSocket endpoints, all of which are vulnerable. Each endpoint lacks both Origin header validation on the upgrade request and any form of authentication.

Terminal I/O: Unauthenticated Command Injection

The terminal I/O endpoint provides raw byte-stream access to the AI agent's pseudo-terminal.

Every message received on the WebSocket is converted to a byte buffer and written directly to the agent's terminal input. There is no Origin check, no authentication, and no input validation. The `taskId` and `workspaceId` parameters are accepted as query string arguments in the WebSocket URL, allowing the attacker to specify which agent session to target.

The function passes the attacker's input through without sanitization. A carriage return byte (`\r`, equivalent to pressing Enter) is interpreted as input submission by the terminal. This means the attacker can construct a complete prompt, append a carriage return, and have the AI agent process it as if the developer typed it.

Runtime State Stream: Unauthenticated Data Exfiltration

The runtime WebSocket endpoint handles upgrade requests in the server's main HTTP handler.

The upgrade handler checks that the request path matches `/api/runtime/ws` and then passes the connection to the runtime state hub. At no point does it examine the Origin header, validate a session token, or perform any authentication check. A connection from `https://attacker.com` is treated the same as a connection from the legitimate kanban UI running on `127.0.0.1:3484`.

Upon successful connection, the server immediately sends a full snapshot of the developer's workspace.

The snapshot contains sensitive information, including absolute filesystem paths to the developer's projects, task titles and descriptions (which may contain proprietary information about features under development), git branch names and commit history (revealing unreleased work), and AI agent chat messages (which may contain code snippets, architectural discussions, or credentials passed to the agent).

After the snapshot, the WebSocket streams live updates as the developer works. An attacker who maintains the connection receives a continuous feed of the developer's activity.

Terminal Control: Unauthenticated Session Termination

The terminal control endpoint accepts structured JSON messages for session management.

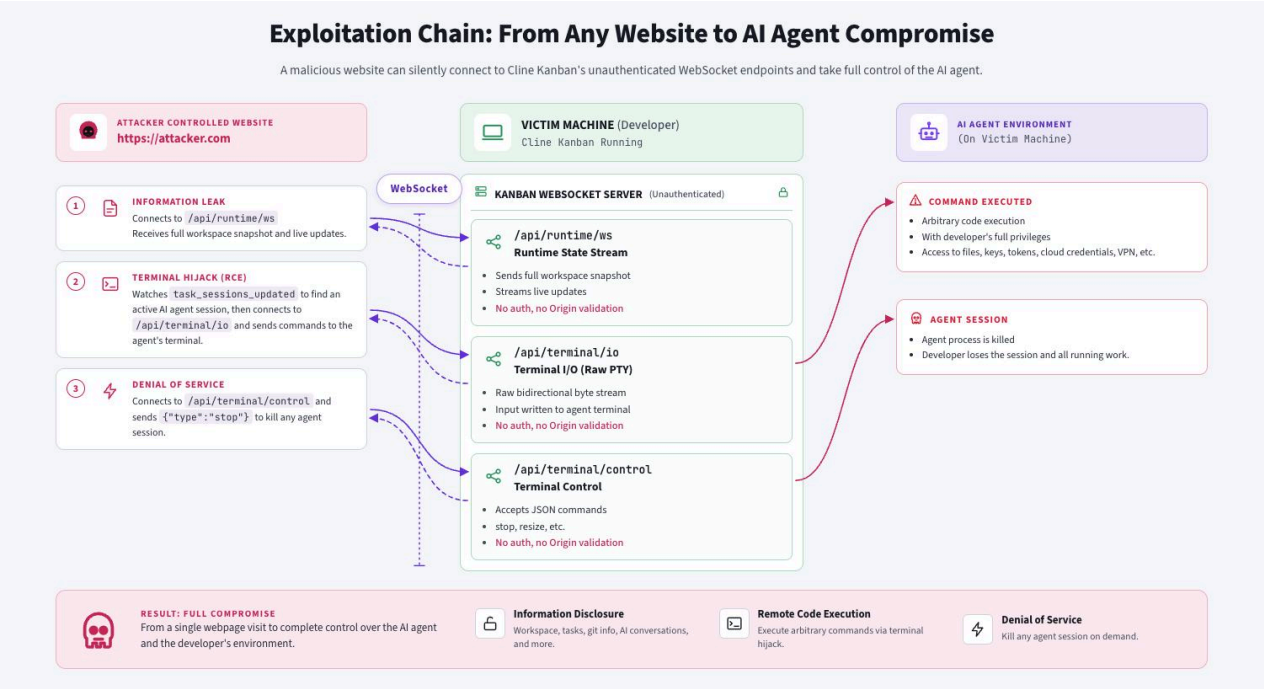
A WebSocket message with `{"type": "stop"}` terminates the specified agent task session. As with the other endpoints, there is no Origin validation, no authentication, and the `taskId` is supplied by the attacker via query parameters.

Root Cause Summary

All three endpoints share the same root cause: the WebSocket upgrade handler and connection handlers do not perform Origin header validation and require no authentication credentials. The server implicitly trusts that any connection to `127.0.0.1:3484` originates from the legitimate kanban UI. As established in Section 3, this assumption is incorrect. Any website can initiate WebSocket connections to localhost, and the browser will not prevent it.

Exploitation Chain

The three unauthenticated endpoints can be chained into a complete attack sequence that progresses from passive reconnaissance to remote code execution. The entire chain executes from JavaScript running on any webpage the developer visits while Cline Kanban is active.



Information Disclosure

A malicious webpage connects to the runtime WebSocket and silently receives a full workspace snapshot, including project paths, tasks, git branches, and agent conversations.
Live updates continue streaming, providing real-time visibility into developer activity.

Session Discovery

The attacker monitors task_sessions_updated messages to identify active agent sessions.
Required identifiers (taskId, workspaceId) are exposed directly by the server.

Remote Code Execution

Using the discovered session, the attacker connects to the terminal WebSocket and injects a command.
The payload is executed by the AI agent as if entered by the user, enabling arbitrary code execution on the developer's machine.

Denial of Service

The attacker can terminate active sessions via the stop endpoint, disrupting the developer's workflow.

Impact Assessment

Impact Summary

Capability	Details
Information Disclosure	Workspace paths, task content, git branches, and AI chat messages are streamed in real-time from any website
Remote Code Execution	Terminal hijack injects commands into the AI agent when a task is active, executing with the developer's full user privileges
Denial of Service	Any running agent task can be killed via the control WebSocket

Scope of Exposure

The vulnerability affects any machine running Cline kanban on any operating system (macOS, Linux, Windows). The attack works across Chrome, Firefox, and Arc. Developers represent a high-value target set with access to source code, CI/CD credentials, cloud provider API keys, SSH keys, and VPN access to internal networks. Compromising a developer's machine provides a foothold for lateral movement into the broader organization.

Stealth Characteristics

The attack is difficult to detect. WebSocket connections to localhost do not appear in browser developer tools unless the developer is actively monitoring network traffic. The kanban server does not log the Origin header of incoming connections. The injected terminal input appears alongside legitimate user input, making it indistinguishable from normal interaction in after-the-fact review.

Comparison with Prior Work

This vulnerability follows the same pattern as the OpenClaw localhost WebSocket exploitation. The consistency of this vulnerability class across independently developed AI agent platforms suggests the risk is systemic. Developers building localhost WebSocket servers for AI agent communication are repeatedly making the same error: relying on the loopback address as a trust boundary without understanding that browsers do not enforce that boundary for WebSocket connections.

Managing AI Agent Access at Scale

The Cline kanban vulnerability is a single instance of a recurring pattern: AI coding agents that expose powerful local interfaces protected by nothing more than the assumption that localhost connections are trustworthy. As this paper and prior research (including OpenClaw findings) demonstrate, that assumption does not hold in a world where developers routinely browse the web while running AI agent tools.

The challenge extends beyond the security of individual tools. Organizations adopting AI coding assistants face a proliferation of agent-local services, each with its own communication model, authentication approach (or lack thereof), and attack surface. The Cline kanban server uses WebSockets on port 3484. Other tools use different ports, different protocols, and different trust models. The organizational security team has limited visibility into which agents are running on developer machines, what capabilities those agents have, and what network interfaces they expose.

Traditional endpoint security tools are not designed for this threat model. EDR systems detect known malware signatures and anomalous process behavior. They are not designed to distinguish between a legitimate AI agent executing a developer-requested shell command and the same agent executing an attacker-injected command delivered via a cross-origin WebSocket hijack. From the endpoint's perspective, both appear to be the agent process spawning a child process. The malicious intent is embedded in the WebSocket message, not in the execution of the process.

This gap between the capabilities of AI agent tools and security teams' ability to govern them is the defining challenge of AI agent security. It manifests in several specific dimensions

- **Unmanaged local interfaces.** AI agents expose HTTP servers, WebSocket endpoints, and IPC channels on developer machines. Security teams have no inventory of these interfaces and no mechanism to enforce authentication requirements.
- **Implicit trust inheritance.** When an AI agent executes a command, it does so with the developer's full user privileges, inheriting SSH keys, cloud credentials, VPN access, and filesystem permissions. A single compromised agent session is equivalent to compromising the developer's account.
- **Opaque agent activity.** Actions an AI agent takes are determined by its model, context, and configuration. There is no predefined set of operations to authorize. Traditional access control models, which operate on predefined resource/action pairs, cannot express policies over open-ended agent behavior.
- **Scale and velocity.** Developer teams adopt AI coding tools rapidly and independently. By the time a security team becomes aware of a new agent tool, it may already be deployed across hundreds of developer machines, each running an unauthenticated local server.

The fix for Cline's kanban server is Origin validation and token-based authentication. But the broader problem persists: every new AI agent tool introduces new local interfaces, a new trust model, and a new attack surface. Managing these at an organizational scale, with consistent policy enforcement, intent-aware access evaluation, and actionable audit trails, requires tooling built specifically for the AI agent era. This is the problem the Oasis Security was built to solve.

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, Craft Ventures, 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 authors

Sagi Layani, Solutions Architect at Oasis Security, LinkedIn

Sagi Layani is a Solution Architect at Oasis Security, where he partners with enterprise security, IAM, and engineering teams to operationalize non-human identity governance and agentic access management across hybrid and multi-cloud environments.
Before joining Oasis, Sagi spent nearly six years as a Lead Software Engineer and R&D Team Leader.