WHITE PAPER

CYTEX™
AI-Powered Resilience Command Center

Claude Code's Network Sandbox Bypass

SOCKS5 Hostname Null-Byte Injection Vulnerability

hello@cytex.io | https://cytex.io

Claude Code Network Sandbox Bypass Persisted Five Months Through 130 Releases

Technical analysis of the SOCKS5 hostname null-byte injection vulnerability in Claude Code's network sandbox, silently patched on April 1, 2026 with no advisory and no CVE.

What Happened

A network sandbox bypass vulnerability existed in Claude Code for approximately five and a half months, present in every release from **v2.0.24 (October 20, 2025) through v2.1.89**. Across roughly **130 published versions** during that window, the vulnerability allowed an attacker who could induce execution inside a sandboxed Claude Code session to bypass the sandbox's network allowlist and exfiltrate credentials, source code, environment variables, and internal infrastructure data to attacker-controlled destinations.

The vulnerability was fixed in **v2.1.90 on April 1, 2026**. The fix shipped silently. No mention of a security correction appears in the release notes. No CVE has been published in the National Vulnerability Database or the GitHub Advisory Database for this issue against Claude Code as of mid-May 2026. The only related CVE, **CVE-2025-66479**, was assigned to sandbox-runtime, the underlying library, not to Claude Code itself.

The vulnerability was disclosed by independent researcher **Aonan Guan**. The associated HackerOne report (#3646509) was closed as a duplicate. The Claude Code security advisories page lists no sandbox vulnerabilities for the affected window.

Vulnerability Profile

Attribute	Detail
Class	SOCKS5 hostname null-byte injection leading to network allowlist bypass
Affected product	Claude Code
Affected versions	v2.0.24 (October 20, 2025) through v2.1.89
Approximate version count	~130 published versions
Vulnerable component	sandbox-runtime <= 0.0.42
Fixed in	Claude Code v2.1.90

Attribute	Detail
Patch date	April 1, 2026
CVE for Claude Code	None published
CVE for sandbox-runtime	CVE-2025-66479
Vendor advisory	None
Discloser	Aonan Guan
HackerOne report	#3646509 (closed as duplicate)

How the Bypass Works

The Claude Code sandbox routes outbound traffic through a **SOCKS5 proxy** that enforces a network allowlist. Customers configure the allowlist with hostname patterns (for example, *.google.com) to permit Claude Code to reach specific destinations while blocking everything else. The intent is to constrain what an agent running inside the sandbox can connect to.

The defect lives in a mismatch between two different layers of string handling:

The validation layer (JavaScript). The proxy uses JavaScript's endsWith() method to check whether a hostname matches the configured allowlist. JavaScript strings handle null bytes (\x00) as ordinary characters. A hostname like attacker-host.com\x00.google.com ends with .google.com from JavaScript's perspective, and the allowlist check returns true.

The resolution layer (libc). When the proxy passes the hostname to the operating system for DNS resolution via getaddrinfo(), libc treats the null byte as a string terminator. Everything after \x00 is discarded. The hostname that actually gets resolved is attacker-host.com.

The validator and the resolver disagree on what the hostname is. The validator approves a connection to what it sees as a .google.com destination. The resolver opens a connection to attacker-host.com. The allowlist is bypassed without modification.

The vulnerable code path in sandbox-runtime <= 0.0.42 passes the raw DOMAINNAME bytes from the SOCKS5 CONNECT request directly into the allowlist matcher with no null-byte rejection, no length cap, and no character allowlist. The single character that turns the allowlist into theater is \x00.

Attack Chain When Paired With Prompt Injection

The sandbox bypass alone is not directly exploitable. It requires an attacker to first get code execution inside the sandbox. In practice, the realistic delivery vector is **prompt injection**, which is now an established class of attack against agentic coding assistants.

The chain:

Step 1. A malicious instruction is planted in content that Claude Code will read during normal operation. Common locations include GitHub issues, README files, documentation pages, commit messages, code comments, dependency manifests, and any other source the user instructs the agent to consume.

Step 2. The user invokes Claude Code on a repository, issue, or task that causes the agent to read the poisoned content. The hidden instruction tells the agent to execute attacker-supplied logic, frequently disguised as a legitimate task.

Step 3. Claude Code executes the attacker-controlled code inside its sandbox. The code uses the null-byte injection to construct SOCKS5 CONNECT requests that pass the allowlist check but resolve to attacker-controlled infrastructure.

Step 4. The sandbox connects to the attacker's destination. Data is exfiltrated over raw SOCKS5, which does not appear in HTTP egress logs and bypasses standard outbound traffic monitoring.

The prompt injection step is what converts a sandbox isolation defect into an exploitable supply chain attack against developer environments. The agent's default behavior of reading external content and following instructions in that content is what makes the bypass reachable without direct attacker access to the developer's machine.

What an Attacker Can Exfiltrate

Once the sandbox is bypassed, the data accessible to the running Claude Code process is available for exfiltration. On a typical developer or CI workstation, that includes:

- **AWS credentials** from ~/.aws/credentials and ~/.aws/config
- **GitHub tokens** from ~/.config/gh/hosts.yml
- **Cloud instance metadata** from 169.254.169.254 on EC2, GCE, and equivalent endpoints, where the developer environment runs inside a cloud instance
- **Internal API endpoints and intranet resources** reachable from the developer's network position
- **Environment variables**, including model API keys, database connection strings, and CI/CD secrets injected into the shell context
- **Source code** from any repository on the local filesystem

Exfiltration runs over raw SOCKS5 traffic to the attacker's destination. Standard HTTP egress logging, web proxy inspection, and DLP tooling configured for HTTP/HTTPS will not see the exfiltration channel. Network-level telemetry that captures SOCKS traffic specifically is required to detect the activity.

The Disclosure Gap

The operational concern around this vulnerability extends beyond the technical defect. The five-and-a-half-month window during which Claude Code shipped a known-vulnerable sandbox closed on April 1, 2026 without:

- A security advisory on the Claude Code security advisories page
- A CVE assigned to Claude Code in the National Vulnerability Database
- A CVE assigned in the GitHub Advisory Database for the Claude Code product
- A mention of the security correction in the v2.1.90 release notes
- Public communication to customers running affected versions

The single CVE that exists for this vulnerability class, **CVE-2025-66479**, was assigned to the underlying sandbox-runtime library. Customers tracking vulnerabilities against the Claude Code product directly would not have surfaced this issue through standard CVE feeds. The associated HackerOne report was closed as a duplicate, suggesting the issue was known internally before the silent patch, though the timeline between internal awareness and the v2.1.90 fix is not publicly documented.

The relevant operational signal for security teams is that the vendor sandbox cannot be treated as the authoritative security boundary for agentic coding tools. **Defense-in-depth is required**, and it must be enforced outside the agent's reach.

Indicators and Detection Signals

The vulnerability does not produce traditional malware indicators. The exploitation uses legitimate SOCKS5 protocol behavior combined with a string-handling mismatch. Detection is behavioral and network-layer focused:

SOCKS5 traffic indicators:

- SOCKS5 CONNECT requests with DOMAINNAME fields containing the byte 0x00 anywhere other than at the end of the string
- SOCKS5 requests with hostname fields longer than valid DNS hostname constraints (greater than 253 characters)
- Outbound SOCKS5 connections from developer workstations or CI runners to destinations outside the configured allowlist

Host-level indicators on potentially-compromised systems:

- Claude Code processes with parent process trees rooted in user invocation against repositories containing untrusted content (GitHub issues, README files, third-party documentation)
- Reads against ~/.aws/credentials, ~/.config/gh/hosts.yml, ~/.ssh/id_*, and environment-variable enumeration from node or claude processes during agent execution
- HTTP requests from Claude Code processes to 169.254.169.254 on cloud-hosted developer environments
- Bulk file reads across source repositories that exceed normal agent operation patterns

Version verification:

- Run claude --version to confirm the installed version
- Versions v2.0.24 through v2.1.89 are vulnerable
- Versions v2.1.90 and later are patched

Defensive Actions

Update Claude Code immediately. Confirm v2.1.90 or later is installed across all developer workstations, build hosts, and CI runners where Claude Code is in use. Run claude --version on each system to verify.

Audit historical SOCKS traffic. Any environment that ran Claude Code with a wildcard or partial-wildcard allowlist on a credential-bearing system between October 20, 2025 and the upgrade date should be assumed exposed. Review SOCKS-mediated outbound traffic logs for connections to destinations outside the legitimate allowlist scope during that window.

Rotate reachable credentials. Treat any AWS, GCP, Azure, GitHub, npm, SSH, Kubernetes, Vault, or other credential present on systems running affected Claude Code versions as potentially compromised. Rotate credentials and audit access logs for unauthorized usage during the exposure window.

Enforce egress controls outside the agent's reach. The structural lesson of this vulnerability is that the vendor sandbox is a software-layer control that runs in the same trust domain as the code it is sandboxing. Egress controls enforced at the network firewall, the cloud security group, or the hypervisor layer cannot be bypassed by a defect in the agent's own validation logic. Network egress allowlists for developer subnets and CI environments should not depend on the agent's internal sandbox for enforcement.

Treat agentic coding tools as a regulated attack surface. The combination of prompt injection susceptibility and credential-rich developer environments makes coding agents a high-value target. Standard endpoint controls, credential vaulting, short-lived credential issuance, and centralized secret management apply with the same rigor as for production servers.

Monitor for additional disclosures. The five-and-a-half-month silent-patch window for this vulnerability suggests that the public CVE feed is an unreliable mirror of the actual security state of agentic coding tools. Subscribe to independent researcher disclosures and security research blogs in addition to vendor advisories.

Sources

- Original disclosure and technical analysis: Aonan Guan, *Second Time, Same Sandbox: Anthropic Claude Code Network Allowlist Bypass and Data Exfiltration*, May 2026