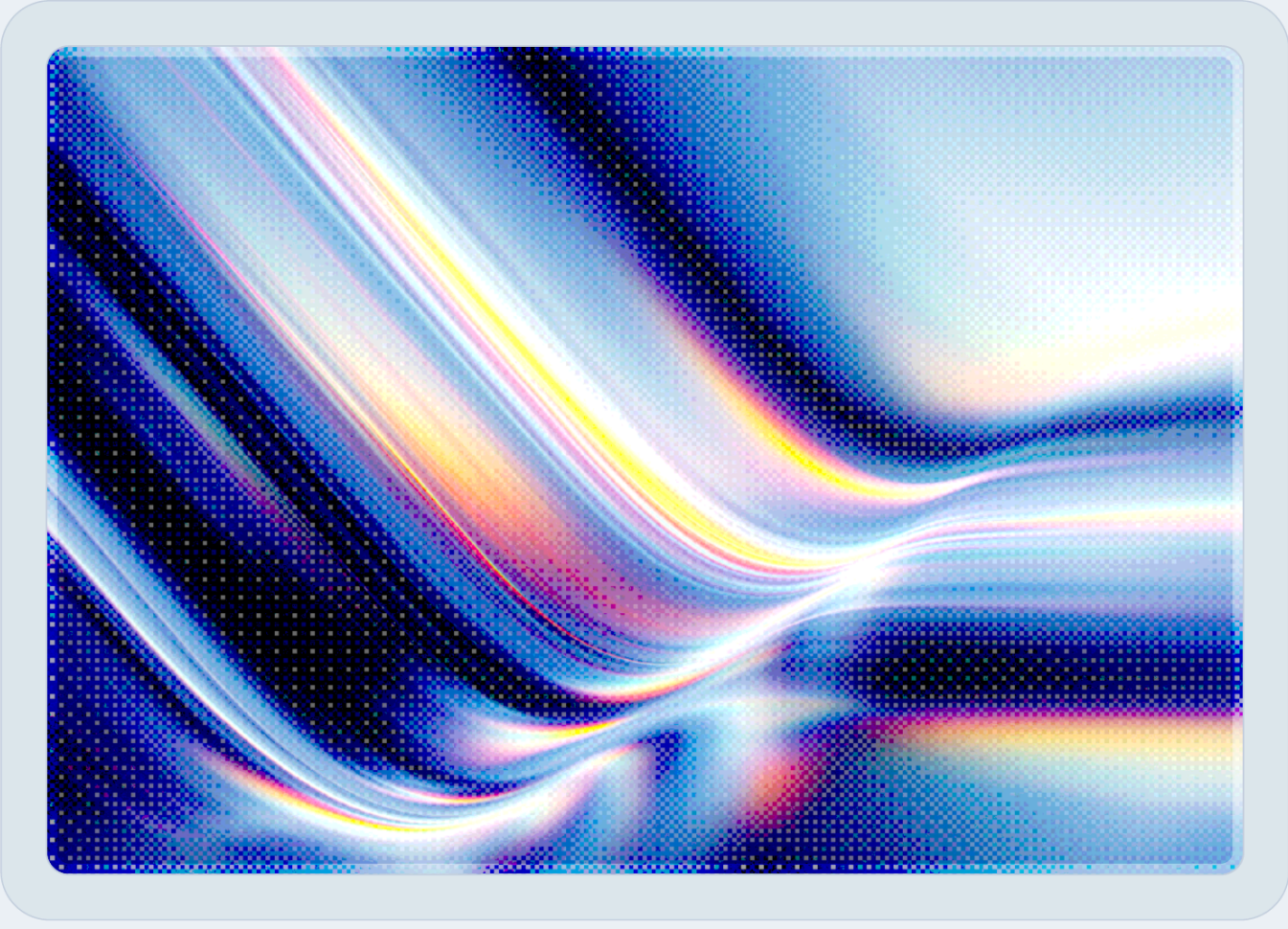
CSAI Foundation | Cloud Security Alliance

Two Codex Sandbox Escapes Reach the Host Machine

2026-09-22

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- Independent security researcher Oren Yomtov of Accomplish AI disclosed two unrelated sandbox-escape flaws in OpenAI's Codex coding agent — Heapjack in Codex Desktop and Overpatch in Codex CLI — both reported to OpenAI on August 12, 2026, and patched within eight days [1][2].
- Heapjack exploited a shared V8 JavaScript memory heap inside Codex Desktop's Node REPL tool, allowing untrusted agent code to read a random validation token out of process memory and impersonate the trusted component that the sandbox's own enforcement logic depended on [1][2].
- The Heapjack attack succeeded even in Codex Desktop's strictest read-only sandbox mode and required no user approval; a victim could trigger it simply by opening a malicious third party's repository and asking Codex a question about the code inside it [1][2].
- Overpatch abused a directory-scoping flaw in the Codex CLI `apply_patch` tool, using a crafted two-entry patch and a symlink to widen a workspace-write grant into unrestricted write access across the local filesystem, enabling persistent code execution through a poisoned shell startup file [1][2].
- OpenAI fixed Heapjack in Codex Desktop build 26.818.21641 and Overpatch in Codex CLI 0.149.0; neither flaw was assigned a public CVE identifier, and both were resolved through direct vendor coordination rather than a formal advisory process [1].
- The underlying failure pattern — a sandbox boundary enforced by a component that shares trust, memory, or a parent process with the code it is supposed to constrain — appears consistent with a recurring architectural weakness CSA has tracked across other AI coding assistants, rather than a Codex-specific defect [3][4].

Background

Codex is OpenAI's agentic coding product line, offered as a desktop application, a browser-based interface, and an open-source command-line tool that developers install locally. Across all three surfaces, Codex is designed to read a developer's repository, propose or apply code changes, and in many configurations execute shell commands on the developer's behalf, which makes the sandbox

wrapped around those executions the primary control standing between a malicious or manipulated task and the host operating system. OpenAI, like most vendors building coding agents in 2026, documents the sandbox as a hard boundary: read-only mode is described as a setting in which the agent cannot write to the filesystem at all, and workspace-write mode is marketed as confinement to the current project directory unless the user explicitly approves an exception.

Yomtov's research, published on the Accomplish AI blog on September 15, 2026, and subsequently reported by BleepingComputer, tested whether those documented guarantees held under adversarial pressure rather than accepting them as given [1][2]. Accomplish AI builds isolation infrastructure for AI coding agents, and Yomtov's disclosure reflects that vantage point: both findings target the mechanism by which Codex decides that a command is trustworthy, not the coding model's behavior or its willingness to follow malicious instructions. He reported both issues to OpenAI on August 12, 2026, through what the company described as a coordinated process, and OpenAI shipped fixes for both within eight days — a response window FourWeekMBA's analysis of the disclosure singled out as a more informative signal of vendor security maturity than the severity of the bugs themselves, since it reflects how quickly an organization can trace an architectural flaw to a specific enforcement point and ship a corrected build [3].

Neither vulnerability received a CVE identifier at the time of disclosure, and OpenAI has not published a standalone security advisory; the only public vendor statement is a spokesperson's confirmation that "we addressed both issues in August... we're continually strengthening our sandboxes" [1]. In CSA's assessment, that absence of a formal advisory does not diminish the severity of what was found. Both flaws allowed code execution outside the sandbox's stated boundary without triggering the approval prompts the product's own documentation describes as the last line of defense, and Heapjack in particular defeated the strictest configuration Codex Desktop offers.

Security Analysis

The two vulnerabilities are structurally distinct, but both stem from the same category of design error: a sandbox that logically depends on the good behavior of code it does not physically isolate. Heapjack targeted Codex Desktop's node_repl component, a Node.js JavaScript execution environment in which OpenAI's trusted orchestration code and the untrusted agent's generated code run inside the same V8 heap rather than in separate, memory-isolated processes. The boundary between those two contexts was enforced entirely by a shared secret — a random token generated fresh on every session — that trusted code was expected to present before the native Rust parent process would execute a request arriving over a shared communication pipe. Yomtov's exploit called the built-in `v8.getHeapSnapshot()` diagnostic function from within the untrusted context, dumped the

process's memory, and scanned the result for strings matching the token's UUID format. Once recovered, the token let the untrusted code assemble a JSON request that was indistinguishable, from the parent process's point of view, from a legitimate command issued by trusted OpenAI code. The parent validated the token and executed the request regardless of which context had actually produced it, ultimately invoking the operating system's native `open` command through launchd — a path that bypassed Codex's own process-tree monitoring — and, on systems with Docker installed, reaching the Docker daemon's Unix socket to gain container-level access as well [1][2].

The consequence of that design is that Heapjack worked at Codex Desktop's strictest configuration, the read-only sandbox mode in which the product is supposed to guarantee the agent cannot write to disk at all, and it required no approval click from the user. Because the trigger condition was simply opening a repository and asking Codex a question about the code inside it, the flaw effectively handed a repository's author the ability to obtain unsandboxed command execution on the machine of anyone who reviewed that repository with Codex Desktop, a scenario with direct relevance to open source maintainers, security researchers triaging pull requests, and any organization that permits Codex to inspect externally sourced code.

Overpatch, by contrast, lived entirely inside Codex CLI's workspace-write mode and exploited the `apply_patch` tool's directory-scoping logic rather than a shared-memory boundary. `apply_patch` grants write access to the parent directory of every file path referenced inside a patch, a convenience meant to let the agent create new files without a separate directory-creation approval step. Yomtov's proof of concept submitted a patch with two entries: the first appended attacker-controlled shell commands to the user's `.zshrc` startup file through a symlink, and the second referenced a path under `/tmp` that never actually executed but existed solely to widen the tool's write grant outward until it encompassed the filesystem root. Once that grant was obtained, later terminal sessions launched by the victim sourced the poisoned `.zshrc` and executed the injected commands automatically and without a sandbox in effect at all, since a fresh shell session sits outside Codex's runtime entirely [1][2].

The following table summarizes how the two flaws compare across the dimensions that matter most for defenders evaluating exposure and prioritizing remediation.

Dimension	Heapjack	Overpatch
Affected surface	Codex Desktop, `node_repl` tool	Codex CLI, `apply_patch` tool in workspace-write mode
Root cause	Shared V8 heap between trusted and untrusted JavaScript contexts; secret-	Directory-scoping logic that grants write access to a patch entry's parent

Dimension	Heapjack	Overpatch
	based (not memory-isolated) trust boundary	directory, chainable to widen scope
Trigger condition	Opening a malicious repository and asking Codex a question about its contents	Applying a crafted patch inside an otherwise normal workspace-write session
Bypassed control	Read-only sandbox mode (strictest available setting)	Workspace directory confinement in write-enabled mode
User approval required	No	No
Execution outcome	Unsandboxed native command execution via launchd; potential Docker daemon access	Persistent code execution via poisoned shell startup file, effectively unrestricted disk write
Fixed in	Codex Desktop build 26.818.21641	Codex CLI 0.149.0

Framed at a higher level, both defects illustrate a distinction that FourWeekMBA's analysis of the disclosure draws explicitly between alignment and containment as separate engineering problems: a coding agent can be faithfully following its instructions and still produce a security incident when the boundary around it is a software convention rather than a structurally enforced wall [3]. In Heapjack's case, the entity doing the enforcing — the parent process validating the token — shared a memory space with the entity it was supposed to be constraining, so the isolation was logical rather than physical. In Overpatch's case, the approval step that was supposed to gate filesystem writes outside the workspace was itself a target the exploit could manipulate rather than a fixed reference point, which is precisely the condition under which, as FourWeekMBA's analysis puts it, "the asking is the part that gets bypassed" [3]. Yomtov's own recommended mitigation reflects this lesson directly: Accomplish's architecture runs coding agents entirely inside virtual machines, treats everything inside the guest — including root access within that guest — as untrusted, and keeps real credentials on the host, issuing the agent only placeholder credentials routed through a proxy it does not control [2].

Recommendations

Immediate Actions

- Confirm that all Codex Desktop installations are running build 26.818.21641 or later and that all Codex CLI installations are running version 0.149.0 or later; treat any earlier version as vulnerable to both disclosed escapes regardless of configured sandbox mode.
- Audit recent Codex Desktop sessions in which a repository from an external, unaffiliated, or otherwise untrusted source was opened, particularly sessions that occurred before the August 20, 2026 fix window, and review endpoint logs for unexpected `open`/launchd activity or Docker socket access correlated with those sessions.
- Review shell startup files (`.zshrc`, `.bashrc`, and equivalents) on developer workstations that used Codex CLI's workspace-write mode prior to the fix for unrecognized appended commands, since Overpatch's persistence mechanism specifically targets these files.

Short-Term Mitigations

- Do not treat a coding agent's advertised sandbox mode — including a "read-only" or "strictest" setting — as sufficient justification for opening untrusted or externally sourced repositories without additional isolation, given that Heapjack defeated Codex Desktop's most restrictive configuration entirely.
- Run coding agents that process untrusted repository content inside an additional layer of host-level isolation, such as a disposable virtual machine or container with no access to production credentials, rather than relying solely on the agent vendor's internal sandbox to be the only enforcement layer.
- Extend code-review and patch-application tooling policies to flag any patch that references paths outside the immediate project directory — including symlinked paths and references to system directories such as `/tmp` or the user's home directory — for manual review before automatic application.

Strategic Considerations

- Incorporate agent sandbox architecture into vendor risk assessments for AI coding tools as a distinct evaluation criterion, asking specifically whether the boundary between trusted and untrusted execution contexts is enforced through physical process or memory isolation rather

than through a shared secret, token, or convention that untrusted code can potentially observe or manipulate.

- Track vendor disclosure-to-fix timelines, such as OpenAI's eight-day resolution of both Heapjack and Overpatch, as an ongoing maturity signal for AI coding agent vendors, since the speed and clarity of a fix reflects the vendor's internal ability to trace an architectural flaw to its enforcement point rather than merely patch a symptom.
- Recognize that Heapjack and Overpatch are the latest instances of a recurring vulnerability class affecting multiple AI coding assistants, not an isolated Codex defect, and update internal AI agent security standards to require evidence of physically isolated sandboxing — rather than vendor marketing claims — before granting a coding agent broad repository or filesystem access.

CSA Resource Alignment

The Heapjack and Overpatch disclosures extend a pattern CSA's AI Safety Initiative has been tracking across multiple AI coding assistants throughout 2026, and two prior CSA research notes map directly onto the specific failure modes disclosed here.

CSA's "AI Coding Agent Sandbox Escapes: The Trust Handoff Flaw", published in July 2026, examined Pillar Security's disclosures across Cursor, Codex CLI, Gemini CLI, and Antigravity and identified what it termed a "trust handoff" pattern: the sandbox itself remains intact, but a trusted component outside the sandbox later reads, runs, or scans something the agent was allowed to write, and that handoff is where the actual escape occurs. Overpatch is close to a textbook instance of this pattern — the sandbox correctly constrained Codex's direct actions, but a shell startup file the agent was permitted to modify was later executed, unsandboxed, by an ordinary terminal session. Organizations that had already implemented that note's recommendation to treat agent-writable configuration and startup files as a distinct, monitored trust boundary — separate from the sandbox's filesystem confinement — would likely have been better positioned to detect Overpatch's persistence mechanism, though no confirmed cases of this have been reported.

What Heapjack adds to this picture is a boundary type CSA's prior coverage had not yet catalogued: an in-memory token rather than a filesystem path. CSA's "GhostApproval: A Shared Symlink Trust-Boundary Flaw in AI Coding Assistants" documented a closely related July 2026 disclosure by Wiz Research showing that six AI coding assistants could be manipulated via Unix symbolic links into writing attacker-controlled content to sensitive files outside the intended workspace, with approval dialogs in several cases displaying decoy filenames rather than the actual write target [5]. Overpatch's use of a symlink to widen `apply_patch`'s write grant beyond the workspace directory is functionally the same

technique GhostApproval catalogued, applied to a different tool. Together, the two disclosures indicate that agent vendors need to audit trust boundaries across every layer where untrusted and trusted code coexist — memory isolation as well as filesystem scoping.

Both incidents map to the AI Controls Matrix (AICM v1.1), particularly its Application & Interface Security and Infrastructure Security domains, which call for enforcement mechanisms that do not depend on the trustworthiness of the workload being constrained [6]. Security teams evaluating Codex or comparable coding agents should treat vendor responses to trust-handoff and symlink-based escapes as a leading indicator of whether a given product's sandbox is architecturally isolated or merely a cooperative convention.

References

[1] BleepingComputer. "Researchers escape OpenAI Codex sandbox to run commands on host." BleepingComputer, September 2026.

[2] Oren Yomtov, Accomplish AI. "Escaping the OpenAI Codex sandbox, twice." Accomplish AI Blog, September 15, 2026.

[3] FourWeekMBA. "OpenAI Codex Sandbox Flaws Named "Heapjack" and "Overpatch" Expose the Containment Gap in Agent Design." FourWeekMBA, September 2026.

[4] Cloud Security Alliance. "AI Coding Agent Sandbox Escapes: The Trust Handoff Flaw." Cloud Security Alliance, July 2026.

[5] Cloud Security Alliance. "GhostApproval: A Shared Symlink Trust-Boundary Flaw in AI Coding Assistants." Cloud Security Alliance, July 2026.

[6] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.