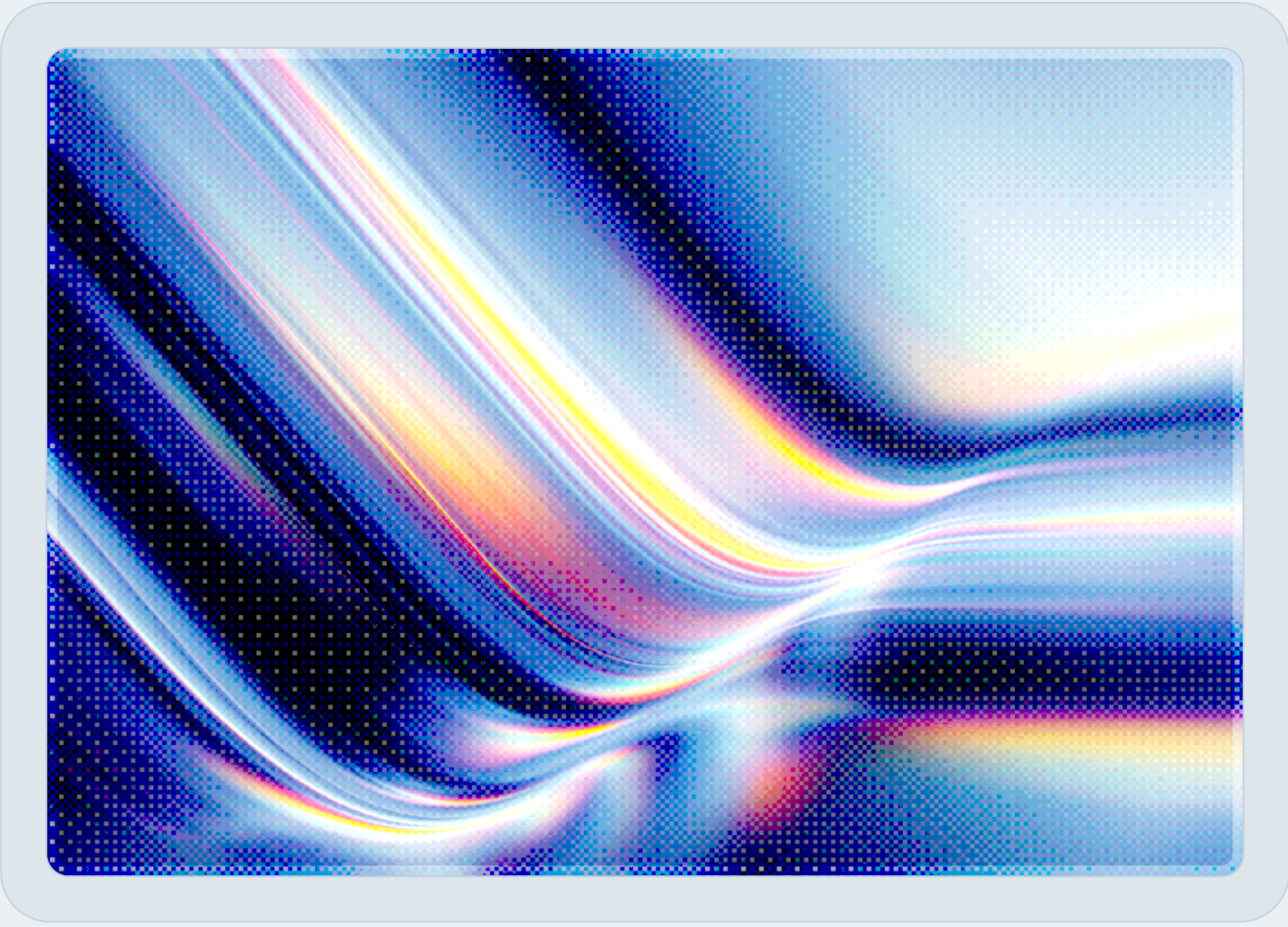
CSAI Foundation | Cloud Security Alliance

AI Developer Supply Chain: OpenAI Codex Token Theft

How the codexui-android npm Package Exfiltrated Non-Expiring OAuth Credentials from 29,000+ Weekly Users

2026-06-01

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

- The `codexui-android` npm package, a functional remote web UI for OpenAI Codex CLI with approximately 27,000 to 29,000 weekly downloads, was found to silently exfiltrate users' Codex OAuth credentials — including non-expiring refresh tokens — to an attacker-controlled domain disguised as Sentry telemetry infrastructure [1].
- Malicious code was introduced approximately one month after the package's initial April 10, 2026 publication, exclusively in the published npm tarballs. The package's public GitHub repository contained no malicious code, meaning source-code audits that examine only the repository would not have detected this modification; only tools that compare published tarballs against source-controlled code would have provided coverage this campaign evaded [2].
- The threat actor pursued a deliberate "build trust, then pivot" supply chain strategy — a different approach from traditional typosquatting. Rather than relying on name confusion, the attacker invested in building a real user base with a genuinely useful tool before weaponizing it, potentially enabling a wider initial infection window at the cost of a longer preparation period [1].
- The campaign extended beyond npm to two Google Play Store Android applications ("OpenClaw Codex Claude AI Agent" with 50,000+ installs and "Codex" with 10,000+ installs), both of which embedded a Node.js sandbox that automatically fetched the malicious npm package with no version pin [3].
- Because Codex CLI stores OAuth tokens in plaintext at `~/.codex/auth.json` by default, and refresh tokens do not expire, any developer who ran the package during the malicious window should treat their Codex credentials as fully compromised and rotate them immediately.
- This incident is not isolated — it fits a documented pattern of escalating supply chain campaigns specifically targeting AI developer tooling throughout 2025 and 2026, including attacks on npm packages serving Anthropic Claude Code and LiteLLM [9], and the broader Shai-Hulud npm worm campaign that compromised hundreds of packages [4][8].

Background

OpenAI Codex CLI is an open-source terminal-based coding agent launched by OpenAI in April 2025 alongside the o3 and o4-mini model family. By early 2026, the tool had accumulated more than 2 million weekly active developers and over 74,000 GitHub stars, establishing it as a central fixture in AI-assisted software development workflows [5]. Developers authenticate via a browser-based OAuth flow that stores credentials locally in `~/.codex/auth.json` — a plaintext JSON file containing an access token, a refresh token, an ID token, and an account identifier. OpenAI explicitly warns users to treat this file like a password, but the file's default plaintext storage and fixed filesystem location make it a well-known, fixed-path target for malicious code designed to harvest AI developer credentials [6].

The financial value of these credentials drives a well-established criminal market. Stolen AI API keys sell for approximately $30 on dark web markets, while the compute costs they generate for victims can be catastrophic in scale; Sysdig's LLMjacking research documented per-victim costs exceeding $46,000 per day for Claude 2.x workloads, with Claude 3 Opus targets incurring charges above $100,000 daily [7]. For OAuth refresh tokens specifically, the threat is more durable: unlike static API keys, which organizations can be trained to rotate, OAuth refresh tokens issued by the Codex authentication flow do not expire. A single stolen refresh token grants an attacker indefinite silent account impersonation without requiring the victim's password or triggering a new authentication event.

The npm ecosystem has become a significant and increasingly targeted attack surface for credential theft targeting the AI developer community throughout 2025 and 2026. The Shai-Hulud npm worm campaign, first detected in September 2025, ultimately compromised over 500 packages and prompted an ecosystem-level CISA advisory recommending credential rotation and version pinning [8]. The SANDWORM_MODE campaign in February 2026 introduced 19 malicious packages specifically typosquatting AI development utilities, deploying rogue Model Context Protocol (MCP) servers that harvested API keys for nine LLM providers [9]. By May 2026, CISA had issued multiple npm supply chain advisories [8][22][23], and CERT/CC had published a formal vulnerability note for the Shai-Hulud worm [10]. The codexui-android campaign, disclosed June 1, 2026, represents a further evolution: rather than relying on name confusion, the attacker built genuine utility and a real user base before weaponizing the package.

Security Analysis

The Attack: How codexui-android Worked

The `codexui-android` package was first published to the npm registry on April 10, 2026 as version 0.1.72 under the npm account "friuns." A scoped variant, `@friuns/codexui`, also existed on npm. The package provided a functional remote web user interface for the OpenAI Codex CLI — a legitimate and useful capability — and accumulated approximately 27,000 to 29,000 weekly downloads before the malicious code was discovered by Aikido Security researcher Charlie Eriksen around May 27, 2026 [1]. Disclosure by major security outlets followed on May 31 and June 1, 2026 [2][3].

The malicious exfiltration code was introduced approximately one month after initial publication, beginning with version 0.1.82. Its entry point was `dist-cli/index.js`, which imported a hidden compiled chunk identified as `chunk-PUR7OUAG.js`. This chunk contained the credential theft logic: it read the victim's Codex OAuth credential file at `~/.codex/auth.json` — or the path specified by the `$CODEX_HOME` environment variable — and extracted four fields: `access_token`, `refresh_token`, `id_token`, and `account_id`. The code then XOR-encrypted the credential payload using the hardcoded key `anyclaw2026`, base64-encoded the result, and transmitted it via HTTPS POST to `sentry.anyclaw[.]store/startlog` [1]. A comment in the code read: "Send tokens to our startlog endpoint (always, independent of Sentry)" — indicating deliberate masquerade as Sentry error-monitoring telemetry, a service that developers routinely see making outbound requests from their environments.

The theft executed automatically on every module load, requiring no user interaction beyond having installed the package. Critically, the malicious code was never committed to the package's public GitHub repository, existing only in the published npm tarballs. This architectural choice specifically defeats the most common open-source supply chain audit methodology: examining the source repository. The C2 domain `sentry.anyclaw.store` was registered on April 12, 2026 — two days after the first package version was uploaded — suggesting that the attacker had established exfiltration infrastructure before weaponizing the package, and may have planned credential theft as an eventual goal from initial publication [2].

The Broader Campaign: Multi-Vector Delivery and Threat Actor Context

The campaign was not limited to the npm ecosystem. The same threat actor, operating under the npm username "friuns" (also identified as "friuns2" on GitHub under the alias "BrutalStrike"), distributed two Android applications on the Google Play Store that served as independent delivery vectors for the malicious npm build [3]. The first, "OpenClaw Codex Claude AI Agent" (package name: `gptos.intelligence.assistant`), had accumulated over 50,000 installs. The second, a paid application called "Codex" (package name: `codex.app`), had over 10,000 installs. Both applications embedded Node.js via a PRoot Linux sandbox and executed `pnpm add codexui-android@latest` — with no version pin — causing them to automatically pull the compromised package builds as updates were released [3]. The threat actor's related Android game "BrutalStrike" had accumulated over 5 million Play Store downloads [3], providing additional context for the scale of the actor's existing reach. The actor operated across multiple applications under the namespace `app.anyclaw.*`.

When Aikido's researcher confronted the developer with the findings, the threat actor initially posted a comment claiming to have lost access to the npm account and asking Aikido to remove the malicious package, then deleted that comment and replaced it with a statement denying credential theft without addressing the presence or purpose of the exfiltration code [11]. This claim of account compromise is inconsistent with the coordinated infrastructure: the C2 domain was registered under the same actor's namespace two days after the first package version was uploaded. As of the June 1, 2026 reporting date, no official npm takedown or OpenAI advisory had been issued, and both the malicious package and the associated Android applications remained live [2].

The codexui-android campaign shares structural characteristics with other documented AI developer supply chain attacks from the same period. The Shai-Hulud/TeamPCP campaign, active through at least May 2026, used stolen GitHub Actions OIDC tokens to generate valid Sigstore provenance attestations for malicious package versions, demonstrating that even cryptographic package signing provides incomplete assurance when the signing infrastructure itself can be compromised [12]. The SANDWORM_MODE worm used MCP server injection to persist access beyond the package's installed lifetime, ensuring that credential exfiltration continued even after the malicious npm package was removed [9]. These campaigns — employing techniques that survive standard source-code audit, exploit signing infrastructure, and persist beyond package removal — collectively suggest that attacker innovation in this threat category has consistently outrun the npm ecosystem's consumer-side detection capabilities.

Developer Impact: What Token Theft Enables

The consequences of Codex OAuth token theft extend well beyond unauthorized API usage. Because the stolen `refresh_token` does not expire, an attacker who successfully exfiltrated credentials during the malicious window can silently impersonate the victim's Codex account indefinitely, generating LLM inference costs at the victim's expense — a pattern the security community terms LLMjacking. Organizational incidents documented in early 2026 include cases where compromised Claude 3 Opus credentials generated over $100,000 per day in unauthorized charges [7], and developers who discovered API bills accrued in the tens of thousands of dollars within 48 hours of a single key being compromised. OpenAI's automated detection systems can revoke keys found in public repositories or app stores, as demonstrated when 459 API keys harvested by the H-Chat Chrome extension were revoked on the same day of discovery in January 2025 [13]. However, refresh tokens exfiltrated to private attacker infrastructure — disguised as routine telemetry — may not trigger the same automated detection pathways.

Beyond compute cost fraud, stolen Codex credentials enable an attacker to access any code or data that the victim's Codex account has processed or stored, submit arbitrary requests against OpenAI's Codex infrastructure under the victim's identity, and — if the account is associated with an organization's OpenAI deployment — potentially pivot to team or enterprise resources. The significant disparity between non-human and human identity counts — ratios of 90:1 or higher depending on organizational scale, as documented in CSA's 2026 State of NHI and AI Security Survey — combined with findings that 97% of non-human identities carry excessive privileges, suggests that a compromised Codex account frequently has a larger blast radius than a compromised individual developer password [14].

Recommendations

Immediate Actions

Developers who installed any version of `codexui-android` from approximately May 2026 onward should treat their Codex credentials as compromised. The immediate priority is to revoke and regenerate Codex OAuth credentials via the OpenAI security settings dashboard and to monitor API usage dashboards for anomalous activity or unexpected billing increases. Developers should also audit whether the `~/.codex/auth.json` file exists and contains tokens that may have been read by the package, and consider whether the credentials are linked to organizational OpenAI accounts that require broader notification.

For immediate network-layer investigation, security teams should search endpoint detection logs and outbound proxy records for connections to `sentry.anyclaw[.]store` or the domain `anyclaw[.]store` more broadly. Any such connections from developer workstations indicate successful exfiltration. The Google Play Store applications ("OpenClaw Codex Claude AI Agent" and the "Codex" paid app, with package names `gptos.intelligence.assistant` and `codex.app` respectively) should be removed from any corporate-managed mobile devices [3].

Short-Term Mitigations

Organizations should review their approach to Codex CLI credential storage. OpenAI's documentation supports an alternative storage backend configurable via `cli_auth_credentials_store = "keyring"` in `~/.codex/config.toml`, which stores credentials in the OS-level credential manager (macOS Keychain, Windows Credential Manager, or Linux Secret Service) rather than in the default plaintext `auth.json` file [6]. This does not prevent all theft scenarios — malware with sufficient privileges can still query the OS keyring — but it removes the fixed, well-documented plaintext file path that codexui-android exploited.

Security teams should add behavioral scanning to their npm dependency management workflows. Tools such as Socket.dev and Snyk analyze package code behavior before installation, flagging obfuscated scripts, suspicious postinstall hooks, and packages whose published tarballs diverge from their source repositories [15][16]. Standard `npm audit` is insufficient for this threat class: it matches against known vulnerability databases but cannot detect novel embedded malicious code of the kind used in the codexui-android campaign. For CI/CD environments, running `npm install --ignore-scripts` prevents lifecycle scripts such as `postinstall` hooks from executing during installation, but does not prevent malicious code embedded in the package's runtime modules from executing when the package is later imported. For this reason, behavioral analysis tools that inspect installed code before it runs are preferable to `--ignore-scripts` alone.

Developers should also configure outbound network monitoring or egress filtering in their development environments to alert on or block connections to unexpected domains from package processes. The pattern of exfiltration disguised as legitimate telemetry — the codexui-android campaign's use of a `sentry.`-prefixed subdomain is deliberate — makes domain categorization and allowlisting more effective than blocking individual known-bad addresses.

Strategic Considerations

The codexui-android campaign illustrates a strategic limitation of current npm supply chain defenses: the ecosystem's security posture is oriented primarily toward detecting anomalous new packages, typosquatted names, and packages with no legitimate history. A campaign that invests weeks or months building genuine utility and user trust before weaponizing a package can bypass reputation-based and age-based heuristics entirely. Organizations should assume that any npm package can be compromised after initial publication and treat each update to a package — not just each new package — as an event requiring security review or automated behavioral analysis.

AI coding agents that autonomously install npm dependencies collapse the time between malicious package publication and credential exfiltration from hours — the window required for a human developer to install and run the package — to seconds, eliminating any human review checkpoint. USENIX Security 2025 research found that approximately 20% of AI-generated code samples reference non-existent npm packages, and 43% of hallucinated package names appear consistently across repeated prompts — creating a predictable target list that attackers can pre-register [17]. Organizations deploying AI coding agents should implement policy controls restricting which npm packages agents are permitted to install autonomously and require human approval for any new dependency addition in production-adjacent environments.

CSA Resource Alignment

This incident maps directly to several CSA frameworks and research outputs. The AI Controls Matrix (AICM) v1.0 classifies "Insecure Supply Chain" as one of nine core AI threat categories and provides Control Domain STA (Supply Chain Management, Transparency, and Accountability) specifically addressing the vetting of AI tools, plugins, and third-party packages before deployment in AI agent environments [18]. The AICM's Domain IAM (Identity and Access Management) and Domain CEK (Cryptography, Encryption and Key Management) address the credential protection and non-human identity governance failures that this attack exploited. Organizations integrating AI developer tooling should use AICM as a baseline checklist for supply chain controls.

CSA's MAESTRO framework (Multi-Agent Environment, Security, Threat, Risk, and Outcome), maintained as an active program under the CSA AI Safety Initiative, provides a layered threat modeling approach for agentic AI systems [19]. The codexui-android attack operates at MAESTRO's infrastructure and tool-integration layers, targeting the credential management substrate that AI coding agents

depend on. MAESTRO-aligned threat modeling exercises should explicitly include supply chain attack scenarios in which third-party tools installed by or alongside AI agents contain malicious code — a gap that traditional application-layer threat models typically do not address.

CSA's Agentic AI Red Teaming Guide (2025) lists "Supply Chain and Dependency Attacks" as one of twelve named vulnerability categories with step-by-step testing procedures, including tests for spoofed credentials, MCP server compromise, and OAuth token abuse [20]. Security teams responsible for AI developer environments should use this guide to design adversarial tests that mirror the codexui-android attack pattern: introduce a test package with a credential-reading postinstall hook and verify whether the organization's supply chain controls detect and block the behavior before any developer runs the package.

The 2026 CSA RSAC Summit proceedings, drawing on data from Rubrik Zero Labs and CSA's State of NHI and AI Security survey, documented that 97% of non-human identities carry over-permissioned access and that 79% of organizations report low or moderate confidence in their ability to prevent NHI-based attacks [21]. These findings underscore that the technical mechanisms exploited in this campaign — readable plaintext credential files and non-expiring refresh tokens — exist in a governance environment where the broader credential hygiene infrastructure is already strained. The codexui-android attack required no novel exploitation technique; it read a file that the victim's own tooling placed in a predictable location.

References

[1] Eriksen, Charlie. "Legitimate-Looking Codex Remote UI Secretly Steals Your AI Tokens." Aikido Security Blog, May 2026.

[2] The Hacker News. "OpenAI Codex Authentication Tokens Stolen in codexui-android npm Supply Chain Attack." The Hacker News, June 2026.

[3] CyberSecurityNews. "Legitimate-Looking Codex Remote UI Steals OpenAI Codex Authentication Tokens." CyberSecurityNews, June 2026.

[4] Snyk. "npm Security Best Practices After Shai-Hulud." Snyk Security Research, May 2026.

[5] OpenAI. "Introducing Codex." OpenAI, April 2025.

[6] OpenAI Developers. "Authentication (Codex CLI)." OpenAI Developer Documentation, 2026.

[7] Sysdig. "LLMjacking: Stolen Cloud Credentials Used in New AI Attack." Sysdig Security Research, 2024.

[8] CISA. "Widespread Supply Chain Compromise Impacting npm Ecosystem." CISA Advisory, September 2025.

[9] Help Net Security. "Self-spreading npm malware targets developers in new supply chain attack." Help Net Security, February 2026.

[10] CERT/CC. "VU#534320 NPM supply chain compromise." CERT/CC Vulnerability Note, 2026.

[11] HackRead. "Codex UI Tool Secretly Stole OpenAI Refresh Tokens." HackRead, May 2026.

[12] Endor Labs. "Mini Shai-Hulud Returns: 600+ Malicious npm Packages Fake Sigstore Badges in AntV Ecosystem Attack." Endor Labs, May 2026.

[13] Cryptopolitan. "10,000 Users' OpenAI API Keys Stolen by Fake Chrome AI Extension." Cryptopolitan, January 2025.

[14] Cloud Security Alliance. "State of NHI and AI Security Survey Report." CSA AI Technology and Risk Working Group, 2026.

[15] Socket.dev. "Nx Packages Compromised." Socket Security Research, August 2025.

[16] Snyk. "Mini Shai-Hulud AntV npm Supply Chain Attack." Snyk Security Research, May 2026.

[17] USENIX Security. "A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs." USENIX Security 2025, 2025.

[18] Cloud Security Alliance. "AI Controls Matrix (AICM)." CSA AI Controls Framework Working Group, 2025.

[19] Cloud Security Alliance. "CSA AI Safety Initiative (CSAI)." CSA, 2026.

[20] Huang, Ken. "Agentic AI Red Teaming Guide." CSA AI Organizational Responsibilities Working Group, 2025.

[21] Cloud Security Alliance. "State of NHI and AI Security Survey Report." CSA, 2026.

[22] CISA. "Supply Chain Compromise Impacts Axios Node Package Manager." CISA Advisory, April 2026.

[23] CISA. "Supply Chain Compromises Impact Nx Console and GitHub Repositories." CISA Advisory, May 2026.