Z E R O T R A C E L A B S

A T T A C K C A S E S T U D Y

Silent Swap: The JavaScript Supply Chain Attack That Weaponized the World's Most Downloaded Packages Against Crypto Users

A single phished developer credential turned billion-download npm packages into multi-chain wallet address hijacking infrastructure.

Attack Snapshot

victim	Multiple JavaScript/npm ecosystem packages affecting Bitcoin, Ethereum, Solana, Tron, Litecoin, and Bitcoin Cash users globally
date	2026-04-06 (discovery date; infection window estimated at 5–12 days prior)
loss	Undisclosed; estimated significant based on package reach of 1B+ weekly downloads
attack Type	Supply chain compromise via developer credential phishing → malicious code injection → clipboard/network-level wallet address replacement
chain	Multi-chain: Bitcoin, Ethereum, Solana, Tron, Litecoin, Bitcoin Cash
attribution	Unattributed at time of publication; operational sophistication and hardware wallet detection logic suggest a well-resourced threat actor, possibly state-affiliated or an established cybercrime syndicate

Executive Summary

On or around April 6, 2026, security researchers identified malicious code injected into widely-used JavaScript packages within the npm registry, collectively downloaded over one billion times per week. The compromise originated from a targeted phishing attack against a prominent open-source maintainer, granting the attacker publish access to

critical upstream dependencies. The injected payload silently monitored outbound transaction data and replaced destination wallet addresses with attacker-controlled addresses across six major blockchains, while incorporating an evasion mechanism that disabled itself upon detection of hardware wallet connections. The incident represents one of the most technically sophisticated and potentially far-reaching supply chain attacks ever directed at the cryptocurrency ecosystem, exploiting the fundamental trust model of open-source package management.

What Happened

The attack began with precision social engineering. A prominent JavaScript developer — a maintainer of packages embedded deep in the dependency trees of countless applications, including crypto wallets, dApps, and DeFi frontends — received a spear-phishing email crafted to impersonate an npm security alert. The email directed the developer to a credential-harvesting page that replicated npm's login portal with near-pixel-perfect accuracy, including a functional two-factor authentication relay (real-time phishing proxy, likely built on an evilginx2-derived framework). The attacker captured both the session token and the OTP, gaining full publish rights to multiple packages under the developer's account.

Within hours of gaining access, the attacker published new minor-version patches to several packages. The version bumps were subtle — semver-compliant patch increments that would be automatically pulled by any project using caret (^) or tilde (~) version ranges in their package.json, which is the default behavior for the overwhelming majority of JavaScript projects. The malicious payload was obfuscated and split across multiple files within the packages, making cursory code review unlikely to catch it. Post-install scripts assembled the payload at build time.

The malware itself was a multi-stage operation. Stage one fingerprinted the host environment to determine whether it was running in a context that handled cryptocurrency transactions — scanning for known wallet application process names, browser extension IDs, and network endpoints associated with blockchain RPC calls. If crypto-relevant context was detected, stage two activated: a network traffic interceptor that performed regex-based pattern matching against wallet addresses for Bitcoin (1/3/bc1 prefixes), Ethereum (0x prefix, 40 hex chars), Solana (base58, 32–44 chars), Tron (T prefix), Litecoin (L/M/ltc1 prefixes), and Bitcoin Cash (q/p prefixes). Matching addresses were silently swapped with attacker-controlled addresses from a rotating pool retrieved from a dead-drop resolver hosted on IPFS, making takedown difficult.

The most operationally notable feature was the hardware wallet evasion logic. When the malware detected USB HID traffic patterns consistent with Ledger, Trezor, or other hardware wallet devices, or identified the presence of hardware wallet bridge software, it suspended address replacement entirely. This was a calculated decision: hardware wallet users are more likely to verify addresses on-device, and a mismatch between the screen display and the device display would immediately expose the attack. By

excluding these users, the attacker maximized dwell time and minimized the probability of early detection.

Discovery was somewhat accidental. On April 6, multiple development teams reported that their CI/CD pipelines began failing due to unexpected post-install script behavior and checksum mismatches in reproducible builds. A security engineer at a mid-size DeFi protocol traced the failures to the updated packages, decompiled the obfuscated payload, and published an emergency advisory. Within hours, npm's security team yanked the affected package versions and revoked the compromised maintainer's tokens. Major crypto security firms, including ZeroTraceLabs, issued simultaneous advisories urging all users to halt transactions from software wallets until they could verify their dependency trees were clean.

The full blast radius remains unknown. Given the packages' download volume — over one billion weekly — the potential number of affected downstream applications is staggering. However, the malware's environmental fingerprinting means only a subset of installations would have activated the payload. Forensic analysis of attacker-controlled wallet addresses revealed incoming transactions across all six targeted chains, though total stolen funds have not been publicly disclosed. Law enforcement investigations are ongoing across multiple jurisdictions.

Kill Chain

1. **Initial Access — Developer Credential Phishing**
   Attacker deployed a real-time phishing proxy (evilginx2-style) targeting a high-value npm package maintainer, capturing session tokens and OTP codes via a spoofed npm security alert email. This bypassed standard 2FA protections and granted full publish access to multiple widely-depended-upon packages.

2. **Payload Injection — Malicious Patch Publication**
   Attacker published semver-compliant patch-version updates to compromised packages containing obfuscated, multi-file malicious payloads triggered via post-install scripts. Caret/tilde version ranges in downstream projects ensured automatic adoption without manual review.

3. **Environment Fingerprinting and Selective Activation**
   Stage one of the payload profiled the host environment for crypto-relevant context: wallet process names, browser extension signatures, and blockchain RPC endpoints. Only qualifying hosts triggered stage two. Hardware wallet connections were detected and excluded to avoid triggering on-device address verification alerts.

4. **Address Replacement and Fund Exfiltration**
   Stage two intercepted outbound network traffic, performed regex matching against wallet address formats for six blockchains, and replaced destination addresses with attacker-controlled addresses sourced from a rotating pool resolved via IPFS-hosted dead drops. Stolen funds were distributed across multiple chains to complicate tracing.

Where Users Failed Themselves

— Blind trust in software wallet address displays without independent verification — users sent transactions to replaced addresses without cross-referencing via a second channel or device.

— Failure to pin exact dependency versions (using lockfiles with integrity hashes) in projects handling financial transactions, allowing automatic ingestion of malicious patch updates.

— Not using hardware wallets for transaction signing — ironically, the attacker's own evasion logic confirms that hardware wallet users with on-device address verification were effectively immune to this attack.

— Developers reusing credentials or relying solely on TOTP-based 2FA rather than phishing-resistant authentication (e.g., hardware security keys with WebAuthn/FIDO2) for package registry accounts with high downstream impact.

— Continued execution of crypto transactions after initial community advisories were published, before verifying local dependency integrity.

Prevention Checklist

FOR INDIVIDUAL USERS

☐ Use a hardware wallet for all non-trivial transactions and always verify the destination address on the device screen before signing — this single practice neutralized this entire attack vector.

☐ Pin exact dependency versions in lockfiles (package-lock.json, yarn.lock) with integrity hashes (sha512) and audit diffs on every update before merging.

☐ Verify destination wallet addresses through an independent out-of-band channel (e.g., QR code from the recipient, separate trusted device) before confirming any transaction.

☐ Run `npm audit` and monitor advisories from Socket.dev, Snyk, and npm security before every build that touches financial infrastructure.

☐ Immediately cease all software-wallet transactions if a supply chain advisory is published for any package in your dependency tree, until a full audit is complete.

FOR PROTOCOLS & PROJECTS

☐ Implement reproducible builds with cryptographic attestation — any CI/CD pipeline producing wallet software or dApp frontends must fail on checksum mismatches, which is exactly what surfaced this incident.

☐ Enforce Subresource Integrity (SRI) or equivalent content hashing for all third-party JavaScript loaded in browser-based wallet and DeFi interfaces.

☐ Mandate hardware security key (FIDO2/WebAuthn) authentication for all maintainers with publish access to packages in the critical dependency path — TOTP and SMS 2FA are insufficient against real-time phishing proxies.

☐ Implement server-side address validation and user-facing address confirmation workflows that do not rely solely on client-side rendering of the destination address.

☐ Adopt a minimal-dependency policy for security-critical code paths; wallet address handling and transaction construction should never traverse third-party packages that are not formally audited.

FOR THE ECOSYSTEM

☐ npm and other registries must enforce mandatory FIDO2/WebAuthn for maintainers of packages exceeding defined download thresholds — the 'critical packages' program needs teeth, not opt-in.

☐ Establish industry-standard provenance attestation for published packages (e.g., npm provenance via Sigstore) and make it a hard requirement for packages used in financial software.

☐ Fund and operationalize continuous automated behavioral analysis of package updates (e.g., Socket.dev's approach) at registry scale, flagging new network calls, post-install scripts, and obfuscation patterns in real time.

☐ Create a coordinated cross-chain rapid response protocol for supply chain incidents affecting wallet software — a standardized mechanism to issue ecosystem-wide transaction hold advisories when upstream dependencies are compromised.

☐ Increase sustained funding for open-source maintainer security — including security training, credential hygiene support, and sponsored hardware security keys — proportional to the downstream financial risk their packages carry.

Key Takeaway

The open-source supply chain is crypto infrastructure, whether the ecosystem acknowledges it or not. A single phished maintainer credential converted billion-download JavaScript packages into a silent, multi-chain fund exfiltration tool — and the only users provably immune were those who verified addresses on a hardware device the malware deliberately chose not to challenge. Trust in software is not a security model.

SOURCES

— ZeroTraceLabs internal threat intelligence analysis — April 2026

— npm Security Advisory (post-incident disclosure) — April 2026

— Socket.dev supply chain threat research briefing — April 2026

— Public advisories and on-chain forensic analysis from multiple blockchain security firms — April 2026

— MITRE ATT&CK: T1195.002 (Supply Chain Compromise: Compromise Software Supply Chain), T1557 (Adversary-in-the-Middle)