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

Ledger Connect Kit Supply Chain Attack: How a Single Compromised npm Account Drained $600K Across the EVM Ecosystem

A phished ex-employee's NPMJS credentials turned a trusted frontend library into a cross-chain wallet drainer.

Attack Snapshot

victim	End users of DeFi frontends consuming @ledgerhq/connect-kit — including SushiSwap, Zapper, Revoke.cash, and dozens of other dApps
date	2023-12-14
loss	$600,000–$700,000 USD in drained assets
attack Type	Supply chain attack via malicious npm package / wallet drainer injection
chain	Ethereum, Polygon, BNB Chain, Avalanche, and multiple EVM-compatible chains
attribution	Unattributed threat actor; likely affiliated with the Angel Drainer kit based on drainer infrastructure observed on-chain

Executive Summary

On December 14, 2023, an attacker phished a former Ledger employee and gained control of their NPMJS account, which retained publish access to the widely-deployed @ledgerhq/connect-kit library. The attacker published malicious versions 1.1.5, 1.1.6, and 1.1.7 containing an injected wallet drainer that silently redirected transaction approvals to an attacker-controlled address when users interacted with any affected dApp frontend. Because connect-kit is a CDN-loaded dependency consumed by dozens of high-traffic DeFi protocols without version pinning, the malicious payload propagated instantly across the ecosystem. Ledger reverted the package within approximately 40 minutes of detection, but the window was sufficient to drain an estimated $600,000–

$700,000 from unsuspecting users — underscoring that frontend supply chain security remains one of the most underinvested attack surfaces in Web3.

What Happened

The @ledgerhq/connect-kit package is a JavaScript library maintained by Ledger that enables dApp frontends to interface with Ledger hardware wallets and various wallet connectors. As a dependency of choice for DeFi frontends, it is consumed by hundreds of projects — many loading it dynamically via CDN rather than bundling a pinned version. This architectural pattern, common for convenience, created a single point of failure that the attacker exploited with precision.

The attack vector was not a vulnerability in Ledger's codebase. It was social engineering. The threat actor targeted a former Ledger employee whose NPMJS account credentials had not been revoked following their departure from the company. Via a phishing campaign — the specifics of which Ledger did not fully disclose — the attacker obtained valid credentials for that account, which still held publish rights to the connect-kit package. This is a textbook offboarding security failure: access lifecycle management on third-party platforms lagged behind internal deprovisioning.

At approximately 12:44 PM UTC on December 14, 2023, the attacker published version 1.1.5 of @ledgerhq/connect-kit to the npm registry. This version contained injected malicious JavaScript that loaded an external wallet drainer payload — assessed to be a variant of the Angel Drainer kit — when executed in a browser context. The drainer intercepted wallet connection events and transaction approval flows, substituting the attacker's address as the recipient. Versions 1.1.6 and 1.1.7 followed in rapid succession, likely to iterate on evasion or delivery. DeFi frontends loading the package via unpinned CDN references — including SushiSwap, Zapper, and others — immediately began serving the malicious code to live users.

User impact was near-instantaneous. Any user who visited an affected dApp, connected their wallet, and approved a transaction during the approximately 40-minute exposure window had their funds redirected. Hardware wallet users were not protected by their device in this scenario: the attack occurred at the frontend layer, meaning the transaction the user was approving on their Ledger device was already the malicious one crafted by the drainer. Blind signing — approving a transaction without fully understanding its parameters — made this attack highly effective.

Ledger's security team identified the compromise at approximately 1:25 PM UTC and published a clean version, 1.1.8, shortly after. They also coordinated with NPMJS to revoke the compromised account's access and worked with Tether to freeze approximately $44,000 in USDT linked to the attacker's address — one of the few concrete post-incident recovery actions. The broader EVM attacker address continued to hold drained assets, and full recovery was not achieved. Ledger subsequently committed to a $600,000 reimbursement to affected users and announced plans to deprecate blind signing support in Ledger devices by June 2024.

The incident is a canonical illustration of transitive trust failure in the npm ecosystem: users trusted dApp frontends, dApp frontends trusted connect-kit, connect-kit trusted its maintainer's credentials — and none of those trust assumptions were enforced with cryptographic or operational controls sufficient to catch a compromised upstream publisher. The blast radius was not bounded by any technical control at the consumption layer.

Kill Chain

1. **Initial Access — Credential Phishing**
   The attacker targeted a former Ledger employee via a phishing attack and successfully obtained their NPMJS account credentials. The account had not been deprovisioned from the @ledgerhq/connect-kit package after the employee's departure, giving the attacker legitimate publish access to a widely consumed library.

2. **Weaponization — Malicious Package Publication**
   The attacker published three successive malicious versions (1.1.5, 1.1.6, 1.1.7) of @ledgerhq/connect-kit to the npm registry. Each version contained injected JavaScript that loaded an external wallet drainer payload — functionally a variant of the Angel Drainer kit — designed to execute in browser environments when the library was initialized by a dApp frontend.

3. **Delivery — CDN Propagation Across DeFi Frontends**
   Dozens of DeFi protocols loaded connect-kit via unpinned CDN references rather than locked dependency versions in their build pipelines. When the malicious versions were published, affected frontends — including SushiSwap and Zapper — immediately began serving the attacker's code to all active users with no deployment action required from the dApps themselves.

4. **Exploitation — Wallet Drainer Execution**
   When users visited affected dApps and initiated wallet interactions, the injected drainer intercepted the transaction approval flow. It replaced legitimate contract call parameters with attacker-controlled values, redirecting asset transfers to the attacker's address. Hardware wallet users were not protected because the malicious transaction was constructed before it reached the signing device.

5. **Actions on Objective — Asset Exfiltration**
   The attacker drained an estimated $600,000–$700,000 in mixed ERC-20 tokens and native assets across Ethereum, Polygon, BNB Chain, and other EVM chains within the ~40-minute exposure window. Drained assets were routed to a single EOA and partially bridged or swapped to complicate tracing. Tether froze approximately $44,000 in USDT on the attacker's address, representing less than 8% of total losses.

Where Users Failed Themselves

— Blind signing on hardware devices: Users approved transactions on their Ledger hardware wallets without verifying the full transaction parameters — destination

address, token contract, and value. Because the malicious frontend had already constructed a fraudulent transaction before presenting it to the device, the hardware wallet's signing step provided no protection against an attacker who controlled the transaction data itself.

— No pre-transaction simulation: Users did not use transaction preview or simulation tools (e.g., Tenderly, Revoke.cash's simulation feature, or wallet-integrated simulators like those in Rabby) before approving. Simulation would have flagged unexpected asset outflows to an unrecognized address.

— Absence of spending limits and token approvals hygiene: Many drained wallets held excessive existing ERC-20 approvals from prior interactions, which the drainer exploited via transferFrom calls without requiring a new approval transaction. Regular approval audits and revocations using tools like Revoke.cash or Etherscan's approval checker would have reduced the drainable attack surface.

— Continuing to interact with dApps after community warnings circulated: Within minutes of the attack becoming apparent, warnings spread across Crypto Twitter and Discord. Users who continued to transact during this window despite public alerts accepted preventable risk.

Prevention Checklist

FOR INDIVIDUAL USERS

☐ Never blind sign: Always verify destination address, token contract address, and value on your hardware wallet's screen before approving. If your wallet or dApp does not surface this data in a human-readable format, treat the transaction as untrusted.

☐ Use transaction simulation before approving: Integrate Rabby Wallet or a browser extension with built-in simulation (e.g., Fire, Pocket Universe) that previews what a transaction will actually do on-chain before you sign it.

☐ Audit and revoke token approvals on a regular cadence — at minimum monthly and always before and after a known incident. Use Revoke.cash or Etherscan's Token Approvals tool. Revoke unlimited approvals to any contract you no longer actively use.

☐ Monitor community channels (DeFi protocol Discords, Crypto Twitter) for incident alerts and halt all dApp interactions during active supply chain incidents until affected protocols issue an all-clear.

FOR PROTOCOLS & PROJECTS

☐ Pin all third-party npm dependencies to exact versions in package-lock.json and enforce integrity verification (npm ci, not npm install). Never load production JavaScript dependencies from CDN without Subresource Integrity (SRI) hashes locked to a specific known-good build.

☐ Implement a formal offboarding checklist that explicitly includes auditing and revoking third-party platform access (npm, GitHub, Vercel, Cloudflare) as a mandatory step when any employee with elevated access departs. Treat npm publish rights as equivalent in sensitivity to production cloud credentials.

☐ Run automated dependency scanning (Snyk, Socket.dev, or GitHub Dependabot with policy enforcement) with alerts on new package versions before they are consumed in production. Socket.dev specifically flags behavioral changes in npm package updates that may indicate supply chain compromise.

☐ Establish a frontend incident response runbook that includes the ability to freeze CDN delivery, revert to a pinned prior version, and notify users within minutes of detecting anomalous dependency behavior.

FOR THE ECOSYSTEM

☐ The npm registry should enforce mandatory 2FA with hardware security keys (not TOTP) for all accounts that have publish access to packages with more than a defined download threshold. Ledger's package had millions of weekly downloads; the compromised account's access should have been protected accordingly.

☐ DeFi aggregators and frontend security tooling providers should maintain real-time monitoring of CDN-loaded dependency hashes and alert consuming protocols automatically when a loaded script hash changes unexpectedly — analogous to certificate transparency monitoring for TLS.

☐ Establish an industry-standard offboarding protocol for Web3 projects that includes automated third-party access audits, given that most teams rely on dozens of SaaS and developer platforms where deprovisioning is entirely manual and frequently neglected.

Key Takeaway

A hardware wallet is only as secure as the code running in the browser that constructs the transaction it signs — compromise the frontend supply chain, and you compromise every user regardless of their device. Access lifecycle management and dependency integrity enforcement are not optional hygiene items; in this incident, the absence of both cost users $600,000 in under 40 minutes.

SOURCES

— Ledger official incident report and post-mortem (December 14, 2023): https://www.ledger.com/blog/a-letter-from-ledger-chairman-ceo-pascal-gauthier-regarding-ledger-connect-kit-exploit

— Ledger Connect Kit npm package audit and malicious version analysis: https://www.npmjs.com/package/@ledgerhq/connect-kit

— Blockaid on-chain analysis of the Angel Drainer infrastructure used in the attack (December 2023)

— Tether USDT freeze confirmation on attacker address: on-chain transaction data, Etherscan address 0x658729879fca881d9526480b82ae00efc54b5c2d

— Socket.dev supply chain security blog — analysis of injected malicious code in connect-kit 1.1.5: https://socket.dev

— SushiSwap and Zapper official incident acknowledgment posts (Twitter/X, December 14, 2023)

— Revoke.cash incident summary and user guidance (December 14, 2023): https://revoke.cash/blog/ledger-connect-kit-exploit