NPM Supply Chain Attack: Comprehensive Analysis & Guidance

Executive Summary

A major supply chain attack has been discovered affecting the NPM ecosystem. The NPM account of a reputable developer known as "Qix" (Josh Junon) was compromised via a phishing email, leading to malicious versions being published for dozens of popular JavaScript packages. These packages collectively have over 1 billion weekly downloads, making this one of the largest supply chain attacks in the JavaScript ecosystem.

The malicious code functions as a sophisticated "crypto-clipper" that steals cryptocurrency by intercepting and modifying transactions, replacing legitimate wallet addresses with attacker-controlled ones.

Date of Discovery: September 8, 2025
Current Status: The maintainer has been notified and is working with the NPM security team. Many malicious versions have been removed, but developers must audit their projects.

Attack Details

Compromise Method

- The developer's account was compromised via a phishing email
- The email appeared to be a legitimate 2FA reset email from "support@npmjs.help"
- The developer confirmed the compromise on Bluesky: "Yep, I've been pwned. 2FA reset email, looked very legitimate."

Affected Packages

The following packages and versions have been identified as compromised:

- ansi-regex@6.2.1
- ansi-styles@6.2.2
- backslash@0.2.1
- chalk@5.6.1
- chalk-template@1.1.1

- color-convert@3.1.1
- color-name@2.0.1
- color-string@2.1.1
- debug@4.4.2
- error-ex@1.3.3
- has-ansi@6.0.1
- is-arrayish@0.3.3
- simple-swizzle@0.2.3
- slice-ansi@7.1.1
- strip-ansi@7.1.1
- supports-color@10.2.1
- supports-hyperlinks@4.1.1
- wrap-ansi@9.0.1

Impact Scale

- Combined weekly downloads of affected packages exceed one billion
- These are foundational dependencies buried deep in dependency trees of countless projects
- Many of the compromised packages are co-maintained with Sindre Sorhus, the most popular maintainer on NPM

Discovery

The attack was initially discovered due to a build error in a CI/CD pipeline:

```
ReferenceError: fetch is not defined
```

This error occurred because the malware was attempting to use the global `fetch` function in an older Node.js environment where it wasn't available. In more modern environments, the attack would have gone unnoticed.

Malware Analysis

Malware Functionality

The malicious code is a sophisticated "crypto-clipper" that uses a two-pronged approach:

1. Passive Address Swapping

- Intercepts all network traffic by monkey-patching `fetch` and `XMLHttpRequest`
- Scans for cryptocurrency wallet addresses in requests and responses

- Uses Levenshtein distance algorithm to find the most visually similar attacker address
- Replaces legitimate addresses with attacker addresses that look very similar
- This makes the swap extremely difficult for users to notice visually

2. Active Transaction Hijacking

- Detects if a wallet extension (like MetaMask) is present
- Patches the wallet's communication methods (`request`, `send`)
- When a user initiates a transaction, it intercepts the data before it's sent to the wallet
- Modifies the transaction to send funds to the attacker's address instead
- The user sees the modified transaction in their wallet and unknowingly approves it

Technical Sophistication

The malware employs several advanced techniques:

1. **Heavy Code Obfuscation**
   - Uses variable names with `_0x` prefixes
   - Employs string encoding and runtime decoding
   - Splits functionality across multiple nested functions
2. **Regex Pattern Matching**
   - Uses sophisticated regex patterns to identify wallet addresses across multiple cryptocurrencies:

```
const obj = Object.entries({
  ethereum: new RegExp(`\\b0x[a-fA-F0-9]{40}\\b`, `g`),
  bitcoinLegacy: new RegExp(`\\b1[a-km-zA-HJ-NP-Z1-9]{25,34}\\b`, `g`),
  bitcoinSegwit: new RegExp(`\\b(3[a-km-zA-HJ-NP-Z1-9]{25,34}|
bc1[qpzry9x8gf2tvdw0s3jn54khce6mua7l]{11,71})\\b`, `g`),
  tron: new RegExp(`((?<!\\w)[T][1-9A-HJ-NP-Za-km-z]{33})`, `g`),
  bch: new RegExp(`bitcoincash:[qp][a-zA-Z0-9]{41}`, `g`),
  ltc: new RegExp(`(?<!\\w)ltc1[qpzry9x8gf2tvdw0s3jn54khce6mua7l]{11,71}\
\b`, `g`),
  ltc2: new RegExp(`(?<!\\w)[mlML][a-km-zA-HJ-NP-Z1-9]{25,34}`, `g`),
  solana: new RegExp(`((?<!\\w)[4-9A-HJ-NP-Za-km-z][1-9A-HJ-NP-Za-km-z]
{32,44})`, `g`),
  solana2: new RegExp(`((?<!\\w)[3][1-9A-HJ-NP-Za-km-z]{35,44})`, `g`),
  solana3: new RegExp(`((?<!\\w)[1][1-9A-HJ-NP-Za-km-z]{35,44})`, `g`),
});
```

3. **Levenshtein Distance Algorithm**
   - Implements the Levenshtein distance algorithm to measure the "edit distance" between strings
   - Uses this to find the attacker-controlled address that is visually most similar to the

victim's address
- This makes the swap extremely difficult to detect visually

4. **DEX Targeting**
   - Specifically targets popular decentralized exchanges:

```
const _0x432d38 = {
  '0x7a250d5630b4cf539739df2c5dacb4c659f2488d': 'Uniswap V2',
  '0x66a9893cC07D91D95644AEDD05D03f95e1dBA8Af': 'Uniswap V2',
  '0xe592427a0aece92de3edee1f18e0157c05861564': 'Uniswap V3',
  '0x10ed43c718714eb63d5aa57b78b54704e256024e': 'PancakeSwap V2',
  '0x13f4ea83d0bd40e75c8222255bc855a974568dd4': 'PancakeSwap V3',
  '0x1111111254eeb25477b68fb85ed929f73a960582': '1inch',
  '0xd9e1ce17f2641f24ae83637ab66a2cca9c378b9f': 'SushiSwap',
}
```

5. **Network Request Interception**
   - Monkey-patches both `fetch` and `XMLHttpRequest` to intercept all network traffic
   - Preserves original HTTP status codes and headers to avoid detection
   - Modifies both request and response data on the fly

Attacker Wallet Addresses

The malware contains numerous attacker-controlled wallet addresses across multiple cryptocurrencies:

Ethereum Addresses (partial list)

```
0xFc4a4858bafef54D1b1d7697bfb5c52F4c166976
0xa29eeFb3f21Dc8FA8bce065Db4f4354AA683c024
0x40C351B989113646bc4e9Dfe66AE66D24fE6Da7B
0x30F895a2C66030795131FB66CBaD6a1f91461731
0x57394449fE8Ee266Ead880D5588E43501cb84cC7
```

Bitcoin Addresses (partial list)

```
1H13VnQJKtT4HjD5ZFKaaiZEetMbG7nDHx
1Li1CRPwjovnGHGPTtcKzy75j37K6n97Rd
1Dk12ey2hKWJctU3V8Akc1oZPo1ndjbnjP
1NBvJqc1GdSb5uuX8vT7sysxtT4LB8GnuY
1Mtv6GsFsbno9XgSGuG6jRXyBYv2tgVhMj
```

Solana Addresses (partial list)

```
5VVyuV5K6c2gMq1zVeQUFAmo8shPZH28MJCVzccrsZG6
98EWM95ct8tBYWroCxXYN9vCgN7NTcR6nUsvCx1mEdLZ
Gs7z9TTJwAKyxN4G3YWPFfDmnUo3ofu8q2QSWfdxtNUt
CTgjc8kegnVqvtVbGZfpP5RHLKnRNikArUYFpVHNebEN
7Nnjyhwsp8ia2W4P37iWAjpRao3Bj9tVZBZRTbBpwXWU
```

Full lists of attacker addresses for all supported cryptocurrencies are available in the malware code.

Protection Guidance

For JavaScript Developers

1. **Check for Compromised Packages**
    - Audit your projects for any of the compromised package versions
    - Check both direct dependencies and transitive dependencies in your lockfiles
2. **Pin Dependencies to Safe Versions**
    - Use the `overrides` feature in your `package.json` to force specific, safe versions:

```
{
  "name": "your-project",
  "version": "1.0.0",
  "overrides": {
    "chalk": "5.3.0",
    "strip-ansi": "7.1.0",
    "color-convert": "2.0.1",
    "color-name": "1.1.4",
    "is-core-module": "2.13.1",
    "error-ex": "1.3.2",
    "has-ansi": "5.0.1"
  }
}
```

3. **Rebuild Your Project**
    - Delete `node_modules` and `package-lock.json`
    - Run `npm install` to generate a new, clean lockfile
4. **Implement Security Best Practices**
    - Use npm audit regularly to check for vulnerabilities
    - Consider using tools like Socket Security or Snyk to detect malicious packages

    - Implement a security policy for dependency management

5. **Monitor for Suspicious Activity**
    - Watch for unexpected network requests in your application
    - Monitor for unusual CPU usage or performance issues
    - Be alert to any cryptocurrency-related anomalies

For Cryptocurrency Users

1. **Use Hardware Wallets**
    - Hardware wallets provide an additional layer of security
    - They show the actual transaction details for verification
    - Always verify addresses on the hardware wallet display
2. **Verify Addresses Carefully**
    - Always double-check cryptocurrency addresses before confirming transactions
    - Compare the entire address, not just the beginning and end
    - Use address book features when available to save trusted addresses
3. **Be Vigilant with Web3 Applications**
    - Be extra cautious when interacting with DeFi platforms and Web3 applications
    - Consider using a dedicated browser or profile for cryptocurrency transactions
    - Regularly clear browser cache and cookies
4. **Monitor Your Transactions**
    - Keep track of all your transactions
    - Set up alerts for unusual activity
    - Regularly check your wallet balances

For Package Maintainers

1. **Secure Your NPM Account**
    - Use strong, unique passwords
    - Enable two-factor authentication
    - Be vigilant about phishing attempts, especially emails about account resets
    - Verify email sender domains carefully (e.g., npmjs.com vs. npmjs.help)
2. **Implement Security Measures for Package Publishing**
    - Consider using signed commits and packages
    - Implement proper access controls for shared packages
    - Regularly audit who has publishing rights to your packages

3. **Monitor Your Packages**
   - Set up alerts for new versions of your packages
   - Regularly check your published packages for unexpected changes
   - Consider using automated tools to monitor for suspicious activity

---

Attack Flow

The attack proceeds through the following steps:

1. **Initial Compromise**
   - Developer receives phishing email appearing to be from NPM support
   - Developer clicks on link and enters credentials
   - Attacker gains access to NPM account
2. **Malicious Package Publication**
   - Attacker publishes new versions of popular packages with malicious code
   - Packages are downloaded by millions of users due to their popularity
3. **Malware Execution**
   - Malicious code runs in user environments
   - Code sets up hooks for network interception
   - Waits for cryptocurrency transactions
4. **Cryptocurrency Theft**
   - When a transaction is detected, malware intercepts it
   - Replaces legitimate wallet address with attacker address
   - Transaction is sent to attacker instead of intended recipient
5. **Evasion**
   - Malware maintains original HTTP status codes and headers
   - Uses visual similarity to make address swaps difficult to detect
   - Preserves application functionality to avoid detection

Lessons Learned

This attack highlights several important lessons for the open-source ecosystem:

1. **Supply Chain Vulnerabilities**
   - The compromise of a single developer account can affect millions of downstream applications

- Popular packages are high-value targets for attackers
- The open-source ecosystem relies heavily on trust

2. **Sophisticated Attack Techniques**
   - The use of Levenshtein distance for visually similar address swapping is particularly clever
   - Attackers are employing increasingly sophisticated techniques to evade detection
   - The combination of phishing and supply chain attacks is particularly effective
3. **Security Awareness**
   - Developers need to be vigilant about phishing attempts
   - Package managers need stronger security measures
   - The community needs better tools for detecting malicious packages
4. **Dependency Management**
   - Pinning dependencies to specific versions is crucial
   - Regular auditing of dependencies is necessary
   - Minimizing dependencies can reduce attack surface

References

1. Anatomy of a Billion-Download NPM Supply-Chain Attack - JD Staerk
2. npm Author Qix Compromised via Phishing Email in Major Supply Chain Attack - Socket Security
3. Tweet by Charles Guillemet (@P3b7_) - Initial disclosure of the attack

---

About This Document

This research and guidance document was compiled based on information available as of September 8, 2025. The situation may evolve as more information becomes available and as the NPM security team and affected maintainers respond to the incident.

Last Updated: September 8, 2025