THREAT INTELLIGENCE REPORT

Scammers Are Not Hackers: 4 Backends Dissected, All Trivially Compromised

Ref. **PD-TI-2026-51239** Published **10 July 2026** Source **phishdestroy.io/scam-infrastructure-exposed**

MULTI-CASE INVESTIGATION
Firebase · Supabase · Express.js · Drainer-as-a-Service · February 2026

~12 min read · Updated **March 2026** · PhishDestroy Intelligence

19,000+
Seed Phrases Stolen (1 campaign)
4
Backends Fully Accessed
0
Authentication Required
267+
Linked Phishing Domains

Disclaimer: Analysis, Not Attack

Everything presented in this article is the result of **passive analysis and publicly accessible data** . No systems were breached. No authentication was bypassed. In each case, the scammers' own

misconfiguration exposed their infrastructure, victim data, and operational identities to anyone who simply looked. Every API key was found in public JavaScript bundles. Every database was wide open by the operators' own design. We document this not to attack — but to demonstrate that the people stealing your crypto **cannot even secure their own tools** .

The Core Thesis: Script Kiddies With Stolen Tools

There is a persistent myth in the public imagination that online scammers are "hackers" — technical masterminds who break into systems with skill and sophistication. **This is wrong.**

Modern crypto scammers are **script kiddies using purchased toolkits** . They buy "Drainer-as-a-Service" packages for $200–$500, deploy them on free or cheap hosting, and pray their victims don't notice. They do not write code. They do not understand networking. They certainly do not understand security.

We know this because in February 2026, PhishDestroy analyzed **4 independent scam operations** — and in every single case, we could have:

- **Read all stolen victim data** (seed phrases, emails, IPs, wallet types)
- **Modified or deleted** the scammer's database
- **Identified the operator** through exposed API keys, email addresses, and infrastructure fingerprints
- **Reproduced the attack against the scammer** — using the same vulnerabilities they left open

No exploits needed. No zero-days. No "hacking." Just **opening the front door they left unlocked** .

Case 1: The Phantom API — server0002.mn19indexpre.xyz

Express.js on Apache, Zero Real Security

NO AUTHENTICATION NO INPUT VALIDATION NO RATE LIMITING CORS: *

Parameter	Value
Domain	`server0002.mn19indexpre.xyz`
IP Address	`108.181.185.225`
Server Stack	Apache/2.4.58 (Ubuntu) → Express.js (Node.js)
SSH Banner	`SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.11`
TLS Issuer	Let's Encrypt (E7), valid Jan–Apr 2026
DNS Registrar	GoDaddy (ns39/ns40.domaincontrol.com)
Email (MX)	`mn19indexpre-xyz.mail.protection.outlook.com`
Microsoft Tenant	`NETORGFT19090185.onmicrosoft.com`
Open Ports	22 (SSH), 80 (HTTP→301), 443 (HTTPS)

"Authentication" — A Joke

The API ships with a hardcoded Bearer token: `thisisakeyforsecureserver` . But here's the punchline — **the token is not actually verified** :

// No auth header → accepted POST / HTTP/1.1 Content-Type: application/json {"subject":"test","domain":"test","messages":["ping"]} → {"success":true} // Wrong token → also accepted Authorization: Bearer wrong_token_completely → {"success":true} // Any content type → also accepted Content-Type: text/xml, text/plain, multipart/form-data → {"success":true}

Zero Input Validation

Every injection payload we tested was silently accepted:

// SQL injection subject: "' OR 1=1--" → accepted subject: "'; DROP TABLE messages;--" → accepted // SSTI (Server-Side Template Injection) subject: "{{7*7}}" → accepted subject: "{{config}}" → accepted subject: "{{self.__class__}}" → accepted // SSRF (Server-Side Request Forgery) domain: "http://169.254.169.254/latest/meta-data/" → accepted domain: "file:///etc/passwd" → accepted // XSS stored payload subject: "<script>alert(1)</script>" → accepted

Performance Fingerprint

Consistent ~7.5 second response time on POST regardless of payload size (10 bytes to 10 MB accepted), suggesting email forwarding or webhook relay on the backend. GET responds in 0.6s. 10 parallel requests complete in 9.1s — no rate limiting enforced.

Deanonymization Potential

Microsoft 365 tenant ID `NETORGFT19090185` is a **direct link to the organization account** that registered this domain. Combined with GoDaddy registration records and a dedicated IP (not behind CDN), this operator is **trivially identifiable** through legal channels. The SPF record (`include:secureserver.net`) confirms GoDaddy email hosting — all email metadata is accessible via subpoena.

Case 2: Firebase Wide Open — web3ledgar.com

Firestore With Zero Security Rules

FIRESTORE RULES: OPEN ALL VICTIM DATA READABLE API KEY IN PUBLIC JS 12 VICTIMS EXPOSED

Parameter	Value
Phishing Domain	`web3ledgar.com` (typosquat of "Ledger")
Alternate Domain	`web3.ledgerscore.ltd`
Firebase Project	`web3ledger-210ab`
API Key	`AIzaSyCv8e-Gl7nK1RPpfNkJt-WjSZiaoe4AsL8`
App ID	`1:1054258933515:web:9fb193fcd0093023f7fc0e`
JS Bundle	`/static/js/main.7a5ec2fa.js`
Firestore Rules	Wide open — unauthenticated read/write
Total Victims	12 records in `users` collection

Collections Found	`users` , `transactions`

Victim Data — Fully Accessible to Anyone

A single unauthenticated GET request to the Firestore REST API returned **every stolen seed phrase** :

GET /v1/projects/web3ledger-210ab/databases/(default)/documents/users?pageSize=300 → 200 OK → Total documents: 12 // Sample victim record (seed phrase redacted for safety) { "walletId" : "W3L-65285520" , "walletType" : "WalletConnect" , "email" : "[REDACTED]@gmail.com" , "seedPhrase" : "[redacted]" , "createdAt" : "2026-02-15T00:14:52.804Z" , "status" : "active" }

Among the 12 records was a telling entry — someone had already tested the system with `fbi@fbi.gov` as the email. The scammer either tested their own system (useful for identification) or someone else had already probed it.

The Attack Flow

The phishing site mimics Ledger's wallet interface. Victim clicks "Connect Wallet" → enters seed phrase → React frontend writes directly to Firestore → scammer reads from the same open database. **No backend server at all.** The entire operation runs on Google's free tier.

Deanonymization Potential

Firebase Project ID `web3ledger-210ab` and App ID `1:1054258933515` are **linked to a Google account** . Google maintains billing records, IP logs, and account creation data for all Firebase projects. A single law enforcement request to Google reveals the operator's identity. Furthermore, the Firestore rules being wide-open means **we could have written records into their database** , notified victims in real-time, or wiped the entire collection.

Case 3: Supabase Full CRUD — web3safe-pal.com

Row-Level Security Disabled, GraphQL Wide Open

RLS DISABLED FULL CRUD ACCESS GRAPHQL ENABLED EDGE FUNCTIONS EXPOSED RUSSIAN LOCALIZATION

Parameter	Value
Phishing Domain	`web3safe-pal.com` (typosquat of "SafePal")
Supabase Project	`gzqsadraigchwdhblavp`
Anon Key	Exposed in `/assets/index-b025f4a6.js` (748 KB)
Database Table	`seeds` — open read, insert, update, delete
GraphQL	Full introspection + mutations enabled
Edge Functions	`send-wallet-import-email` , `send-email`
Email Service	Resend API (RESEND_API_KEY in env)
Victim Records	IDs 130–131 (129 previously deleted)

UI Language	Russian ("Основной кошелек", "Ваш кошелек загружается...")

Full Database Access — Read, Write, Delete

The Supabase anon key, found in the minified JavaScript bundle, grants **complete CRUD access** to the `seeds` table:

// READ — get all stolen seed phrases GET /rest/v1/seeds?select=*&order=id.asc → 200 OK [{ "id" : 130 , "phrase" : "[redacted]" , "name" : "Основной кошелек" }, { "id" : 131 , "phrase" : "[redacted]" , "name" : "Основной кошелек" }] // INSERT — we can write to the scammer's database POST /rest/v1/seeds {"phrase":"PHISHDESTROY_WAS_HERE","name":"test"} → 201 Created { "id" : 132 } // UPDATE via GraphQL mutation { updateseedsCollection(filter:{id:{eq:131}}, set:{name:"PhishDestroy was here"}) { affectedCount } } → {"affectedCount": 1} // DELETE via GraphQL mutation { deleteFromseedsCollection(filter:{id:{eq:131}}, atMost:1) { affectedCount } } → {"affectedCount": 1}
IDs start at 130 — meaning records 1–129 were previously deleted by the operator. At least 131 seed phrases have passed through this system.

Edge Functions: The Email Trail

Two Supabase Edge Functions are active. We reverse-engineered the `send-email` function's expected input format by testing payloads:

// Probing send-email endpoint {"seed":"test phrase","name":"test"} → 200 OK {"id":"f2749592-..."} {"seed_phrase":"...","wallet_name":"..."} → 400 "No seed data provided." {"phrase":"...","name":"..."} → 400 "No seed data provided." // The function accepts {seed, name} and sends to attacker via Resend API // Edge Function source (reconstructed from JS bundle): // NB.functions.invoke("send-wallet-import-email", // {body: {wallet_name, secret_phrase, ip_address}})
The Resend API key (`RESEND_API_KEY`) is stored in Supabase environment variables. Resend maintains sender verification records and billing data — **another direct path to the operator's identity** .

Deanonymization Potential

The Russian UI localization ("Основной кошелек", "Ваш кошелек загружается...") indicates a **Russian-speaking operator** . The Supabase project (`gzqsadraigchwdhblavp`) is linked to an account with billing records. The Resend email service has the recipient email address. The GraphQL introspection reveals the complete database schema. We demonstrated full write access — **we could have replaced every stolen seed phrase with a warning message to victims** , or deleted the entire table. The operator would have no way to recover the data.

Case 4: Industrial-Scale Drainer — aipolypredictor.xyz

19,000 Seed Phrases in 5.8 Days

19K+ SEEDS STOLEN SEQUENTIAL JOB IDs NO CORS RESTRICTIONS DaaS KIT (defex.cc) 11 DOMAINS CONFIRMED

Parameter	Value

Frontend Domain	`aipolypredictor.xyz` ("PolySniper | Frontrun Insider Bets")
C2 API	`api.yfhikblkhghdyteiuyf54.run`
C2 IPs	`172.67.168.147` , `104.21.26.231` (Cloudflare)
Backend	Express.js (Node.js) v1.0.0
Registrar (Frontend)	NiceNIC International Group Co.
Registrar (C2)	PDR Ltd. (PublicDomainRegistry.com)
Uptime	~139 hours (started ~2026-02-10 21:00 UTC)
Drainer Kit CDN	`renderer-postcard.defex.cc` (601 KB obfuscated JS)
Telegram Bot	Active, integrated for notifications
Rate Limit	10 req/60s (only limit found)

Scale Exposed by Sequential IDs

The most damning mistake: **sequential job IDs** . Every seed phrase submission returns an incrementing ID, allowing anyone to calculate total volume:

POST /api/check-seed-full { "seedPhrase" : "test phrase here" , "bundleId" : "88ef78f56e0837dd0339e40a882bf563" , "depth" : 100 , "domain" : "aipolypredictor.xyz" , "sourceInfo" : { "walletName" : "MetaMask" , "isBot" : false } } → {"success":true, "message":"Check queued", "jobId":"19358"} // Next request: jobId 19359, then 19360... // ~19,000 seed phrases in ~139 hours = ~137 per hour

Multi-Chain Architecture

The C2 server derives keys across all major chains using depth-100 derivation paths:

⛓️
Chains Targeted
ETH/BTC/SOL/XMR
🔑
Derivation Depth
100
🌐
Confirmed Domains
11
🕸️
defex.cc Linked
255+

Campaign Infrastructure

11 confirmed domains across 2 operator groups, tracked by Bundle IDs:

Bundle ID	Domains	Status
`88ef78f5...`	aipolypredictor.xyz	LIVE

`4446ea5d...`	solana.onspace.app, solxjup.onspace.app	LIVE
defex.cc kit	jup-v2.com, events-charizard.fun, events-llquid.fun, events-blackswan.fun, soljup.onspace.build	Mixed

JavaScript Payload Analysis

Three obfuscated JS payloads serve the drainer:

- **wallet-connect.js** (46 KB) — Handles wallet connection UI, intercepts seed input. String array rotation obfuscation.
- **wallet-specific-modals.js** (134 KB) — Contains **full BIP39 English wordlist** and **Monero wordlist** (1,626 words). Anti-debugging via console.log/trace overrides. Multi-wallet modal support.
- **defex.cc/index.js** (601 KB) — Unicode-obfuscated with Chinese variable names. Solana-specific drainer logic. Base58 encoder, crypto key derivation primitives. Version 3.0.0.

Deanonymization Potential

The `CORS: *` header and lack of authentication means **anyone can submit requests and observe job ID increments** in real-time. The Telegram bot integration means the operator's Telegram account receives notifications — and Telegram metadata can be subpoenaed. NiceNIC (registrar for the frontend) is a known bulletproof registrar we've previously investigated , but PDR Ltd. (C2 domain registrar) **does respond to law enforcement requests** . The `defex.cc` drainer kit serves 255+ domains — compromising defex.cc would expose the entire DaaS operation and all its customers.

Side-by-Side: 4 Operations, Same Pattern

Metric	mn19indexpre Express.js	web3ledgar Firebase	web3safe-pal Supabase	aipolypredictor Drainer C2
Authentication	None (token ignored)	None	Anon key in JS	None (CORS: *)
Data Readable	Messages/relay	All seed phrases	All seed phrases	Job IDs / scale
Data Writable	Yes (unlimited)	Yes	Yes (full CRUD)	Yes (submit)
Input Validation	Zero	Zero	Zero	Minimal
Rate Limiting	None	None	None	10 req/60s
Deanonymizable	MS365 tenant	Google account	Resend + Supabase billing	PDR + Telegram
Estimated Victims	Unknown	12	131+	19,000+
Operator Language	Unknown	English	Russian	Unknown

Why Scammers Are Not Hackers

The evidence is overwhelming. Across all 4 operations, we observe the same pattern:

What Scammers Do

- Buy pre-made drainer kits ($200–$500)
- Deploy on free tiers (Firebase, Supabase)

- Leave default configurations unchanged
- Use hardcoded tokens they don't verify
- Never enable RLS, never restrict CORS
- Expose their own identities in metadata
- Use sequential IDs that reveal their scale

What Hackers Would Do

- Write custom exfiltration tools
- Use encrypted, authenticated channels
- Randomize identifiers, rotate infrastructure
- Implement proper access control
- Use Tor/proxy chains, anonymous payments
- Separate operational identity from hosting
- Implement anti-forensic techniques

The average Drainer-as-a-Service customer is a **social engineer with a credit card** , not a technical operator. They know how to register a domain and paste code into a hosting panel. They do not know how to:

- Configure Firestore security rules (would take 2 minutes)
- Enable Supabase Row-Level Security (would take 5 minutes)
- Validate and sanitize input (would take 30 minutes)
- Use UUIDs instead of sequential integers (would take 1 line of code)
- Restrict CORS to their own domains (would take 1 line of config)

These are not sophisticated adversaries. **These are people who cannot configure a database.**

Indicators of Compromise (IOCs)

Domains

Case 1: Express.js API server0002.mn19indexpre.xyz # Case 2: Firebase Stealer web3ledgar.com web3.ledgerscore.ltd # Case 3: Supabase Stealer web3safe-pal.com # Case 4: Drainer Campaign aipolypredictor.xyz api.yfhikblkhghdyteiuyf54.run yfhikblkhghdyteiuyf54.run renderer-postcard.defex.cc solana.onspace.app solxjup.onspace.app jupag.onspace.app jupiverse.onspace.app stakepayment.icu jup-v2.com events-charizard.fun events-llquid.fun events-blackswan.fun soljup.onspace.build

IP Addresses

108.181.185.225 # Case 1 — Express.js (dedicated) 172.67.168.147 / 104.21.26.231 # Case 4 — C2 API (Cloudflare) 172.67.178.251 / 104.21.67.171 # Case 4 — Frontend (Cloudflare) 172.67.209.39 / 104.21.37.139 # Case 4 — defex.cc (Cloudflare) 43.130.171.152 / 43.130.171.225 # Case 4 — onspace domains (Tencent)

API Keys & Project IDs

Firebase (Case 2) Project: web3ledger-210ab API Key: AIzaSyCv8e-Gl7nK1RPpfNkJt-WjSZiaoe4AsL8 App ID: 1:1054258933515:web:9fb193fcd0093023f7fc0e # Supabase (Case 3) Project: gzqsadraigchwdhblavp Anon Key: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9... # Campaign Tracking (Case 4) Bundle 1: 88ef78f56e0837dd0339e40a882bf563 Bundle 2: 4446ea5ddb308b2494db8ad4b12196c3 # Microsoft Tenant (Case 1) NETORGFT19090185.onmicrosoft.com

Conclusion: The Emperor Has No Clothes

The Takeaway

Every scam operation we analyzed could be **fully compromised, deanonymized, and disrupted** using nothing more than a web browser, curl, and publicly available documentation. In every case, the attackers left their databases wide open, their API keys in public JavaScript files, their identities in metadata, and their victims' data accessible to anyone who bothered to look.

The lesson is simple: **scammers are not hackers** . They are shoplifters who bought a lock-pick set from AliExpress and forgot to lock their own front door. The tools they use are sophisticated — because someone else built them. The operators themselves are amateurs who reliably expose their own infrastructure, their victims' data, and their own identities to anyone with basic technical literacy.

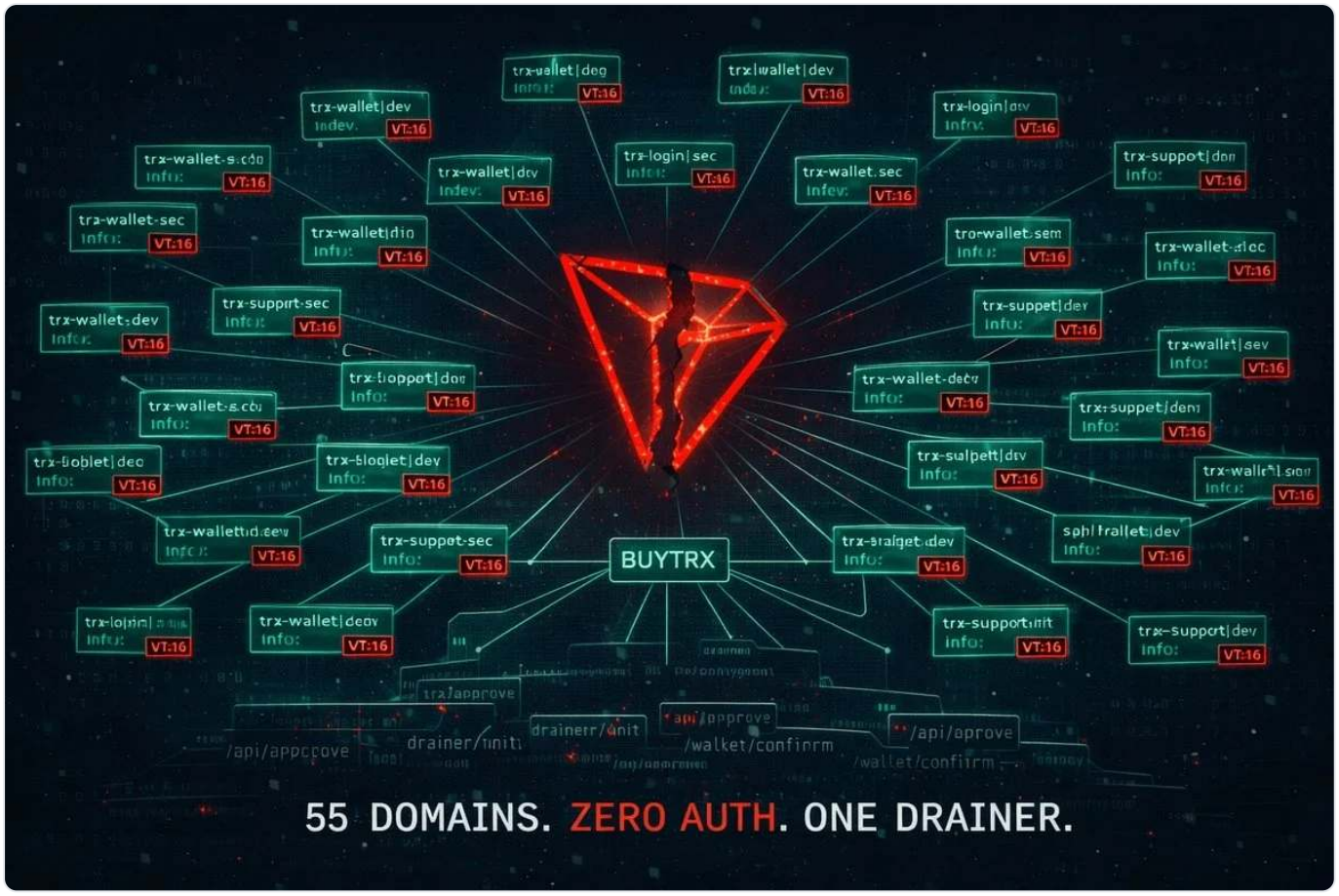
If you've entered your seed phrase on any of these sites — assume your wallet is compromised and transfer funds immediately.

All findings have been reported to the relevant service providers (Google/Firebase, Supabase, Cloudflare, domain registrars) and documented for law enforcement. The IOCs above have been added to the PhishDestroy destroylist .

Share This Investigation

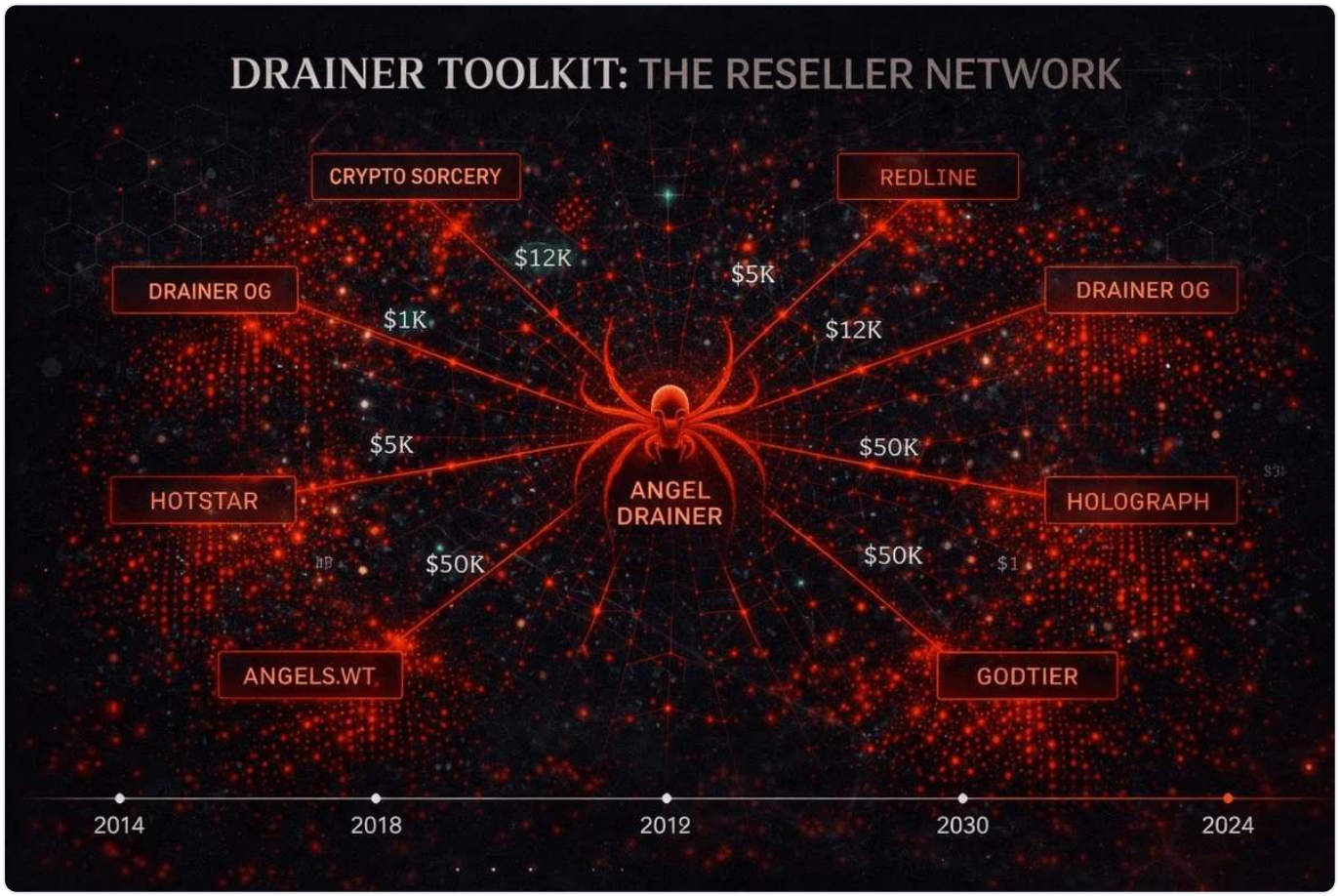
X / Twitter Telegram Reddit LinkedIn

Related Investigations



INVESTIGATION

BUYTRX Exposed: 55 Domains & TRON Approval Drainer



DEEP INVESTIGATION

Crypto Drainer Toolkit: Angel Drainer Resellers Exposed

INVESTIGATION

Keitaro TDS: 1,500 Panels Exposed, Zero Legit Uses