When Git History Lies: Commit-Date Spoofing as Malware Cover

kl4r10n.tech/blog/when-git-history-lies

KL4R10N

Four public GitHub repositories show a shared obfuscated stage-0 JavaScript loader appended to executable config files, aligning with publicly documented XCTDH / DEV#POPPER infrastructure and tactics.

Malware Analysis

Threat Intel

Executive summary

This research article documents four public GitHub repositories that contain an appended, obfuscated JavaScript loader embedded in framework or build-tool configuration files. In all four cases, the malicious code is appended **after** an otherwise legitimate configuration export, allowing the file to remain visually plausible while still executing arbitrary code when the associated tooling loads the config.

Across the four samples, direct inspection shows the same stage-0 scaffold:

- a visible version tag in the form `global['_V']='...'`
- `global['r']=require`

- a custom deobfuscation function named `MDy`
- a large scrambled string blob
- an immediately invoked function expression (IIFE)
- the same general stage-0 execution structure, with only the `_V` tag changing between samples

Public reporting by Ransom-ISAC describes the same stage-0 loader family as **Cross-Chain TxDataHiding (XCTDH)** and says it retrieves payload pointers from TRON or Aptos, fetches payload bodies from BSC transaction input via `eth_getTransactionByHash`, XOR-decrypts the result, then executes one payload inline and launches another as a detached background process. Source: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/

Public reporting also already documents the exact stage-0 infrastructure values associated with this loader family, including the TRON addresses, Aptos addresses, BSC transaction hashes, BSC wallet, and XOR keys listed later in this article. Source: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/

Public reporting further links the broader campaign to **DEV#POPPER** malware variants and describes DPRK-linked / North Korean state-sponsored overlap. Ransom-ISAC describes the campaign as DPRK-linked, while eSentire says it attributes the related threat activity with high confidence to a North Korean state-sponsored APT group. Sources: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/ and https://www.esentire.com/blog/north-korean-apt-malware-analysis-dev-popper-rat-and-omnistealer-everyday-im-shufflin

New tactic spotlight: timeline anomaly via commit-date spoofing

One of the public samples, `ReactCosmosDelegationUI`, has a commit dated **2019-07-08** that already contains the appended obfuscated loader. Commit URL: hxxps://github[.]com/Staking-Facilities-GmbH/ReactCosmosDelegationUI/commit/06931046a0086c02b95a86713ce90683a54d08fd

That timestamp should not be treated as trustworthy proof that the malicious content genuinely existed in 2019.

Two concrete reasons:

1. **Git allows author and committer dates to be set via environment variables.** The official Git documentation explicitly documents `GIT_AUTHOR_DATE` and `GIT_COMMITTER_DATE`. Sources: https://git-scm.com/docs/git-commit/2.35.0 and https://git-scm.com/docs/git-commit-tree/2.25.1.html
2. **The loader family uses Aptos mainnet infrastructure in public reporting, but Aptos Labs' own timeline says Aptos Mainnet launched in October 2022.** Source: https://aptoslabs.com/

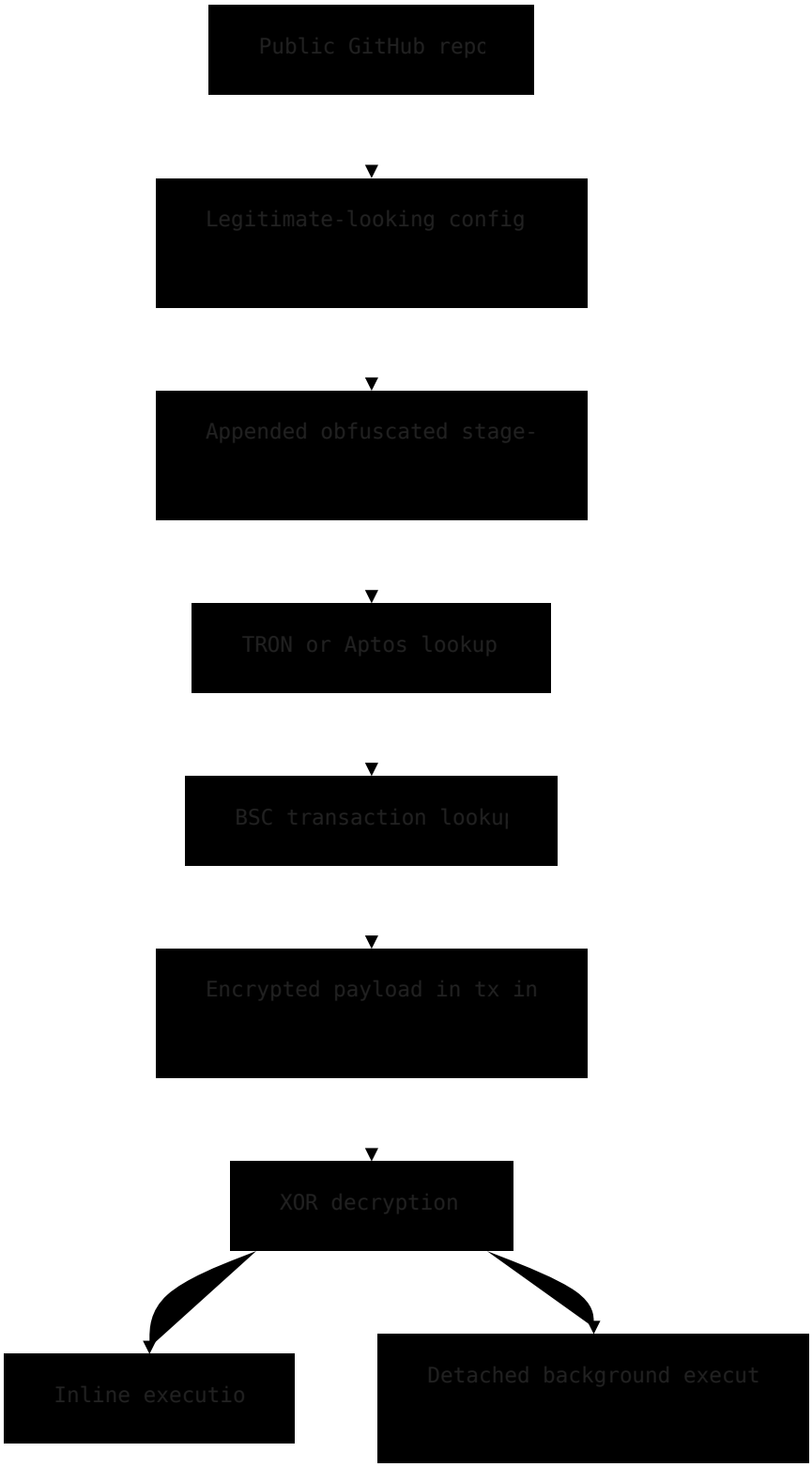
For a practical demonstration of commit-date spoofing mechanics, see:
https://github.com/pcaversaccio/test-spoof-commit-date
Thanks to @pcaversaccio for publicly building and validating this technique.

The most cautious conclusion is therefore:

- the **content** is more informative than the displayed commit date,
- the date is **compatible with timestamp manipulation or rewritten history**, and

- the “2019” appearance should not be used as an argument that the sample is benign.

Scope and methodology



Sample set examined directly

1. `Staking-Facilities-GmbH/ReactCosmosDelegationUI` → `nwb.config.js`
   URL: hxxps://github[.]com/Staking-Facilities-GmbH/ReactCosmosDelegationUI/blob/HEAD/nwb.config.js
2. `subramanianv/CertificateVerification` → `truffle.js`
   URL: hxxps://github[.]com/subramanianv/CertificateVerification/blob/HEAD/truffle.js
3. `scholtz/covid-sk` → `vue.config.js`
   URL: hxxps://github[.]com/scholtz/covid-sk/blob/HEAD/vue.config.js
4. `scads-io/frontend` → `next.config.js`
   URL: hxxps://github[.]com/scads-io/frontend/blob/main/next.config.js

What this article does and does not claim

- This article **does** make direct claims about the code visible in the four GitHub files listed above.
- This article **does** rely on public reporting for the stage-1 / stage-2 blockchain retrieval details and the broader cluster-level IOC set.
- This article **does not** claim that the public blockchain RPC or API domains are inherently malicious. Public reporting and official vendor documentation both indicate those services are legitimate infrastructure that can be abused by attackers. Sources: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/ and https://developers.tron.network/docs/trongrid

Direct sample observations

1) `ReactCosmosDelegationUI` (`nwb.config.js`)

The file begins as a plausible `nwb` configuration object and then appends a one-line obfuscated JavaScript loader after the closing `};`. The visible variant tag in this sample is `8-st21`.
Sample URL: hxxps://github[.]com/Staking-Facilities-GmbH/ReactCosmosDelegationUI/blob/HEAD/nwb.config.js

The repository's public commit history also shows a commit titled **"changed dependency"** with a timestamp of **2019-07-08** that includes the appended loader. Commit URL: hxxps://github[.]com/Staking-Facilities-GmbH/ReactCosmosDelegationUI/commit/06931046a0086c02b95a86713ce90683a54d08fd

2) `CertificateVerification` (`truffle.js`)

The file begins as a plausible Truffle configuration and then appends the same obfuscated loader family after the closing `};`. The visible variant tag in this sample is `8-st57`.
Sample URL: hxxps://github[.]com/subramanianv/CertificateVerification/blob/HEAD/truffle.js

3) `covid-sk` (`vue.config.js`)

The file begins as a plausible Vue CLI configuration and then appends the same obfuscated loader family after the closing `};`. The visible variant tag in this sample is `8-st46`.
Sample URL: hxxps://github[.]com/scholtz/covid-sk/blob/HEAD/vue.config.js

4) `scads-io/frontend` (`next.config.js`)

The file begins as a plausible Next.js configuration and then appends the same obfuscated loader family after the legitimate export line. The visible variant tag in this sample is `8-778`.
Sample URL: hxxps://github[.]com/scads-io/frontend/blob/main/next.config.js

Why these files are execution points, not passive metadata

This matters because the malicious code is not hidden in a random source file; it is embedded in files that the project tooling treats as executable configuration.

- Next.js states that `next.config.js` is a **regular Node.js module**, not JSON, and that it is used by the Next.js server and build phases. Source: https://nextjs.org/docs/pages/api-reference/config/next-config-js
- Vue CLI states that `vue.config.js` is **automatically loaded** by `@vue/cli-service` if it is present in the project root. Source: https://cli.vuejs.org/config/
- Truffle states that `truffle-config.js` is a **JavaScript file** and **can execute any code necessary** to create the configuration. The same documentation also mentions historical `truffle.js` naming behavior on Windows. Source: https://archive.trufflesuite.com/docs/truffle/reference/configuration/
- The `nwb` package documents `nwb`-based development and build scripts and documents `nwb.config.js` as the place where generated configuration can be customized. Source: https://www.npmjs.com/package/nwb

The practical consequence is simple: for a developer to trigger the malicious loader, they do not need to run a suspicious standalone script. Running normal project commands such as build, dev, or test is enough if those commands load the config file.

Shared stage-0 structure across the four samples

Direct comparison of the four config-file payloads shows that they are the same stage-0 family with small per-sample changes.

Shared visible traits

- identical `global['r']=require` pattern
- identical `MDy` deobfuscation function name
- identical general scrambled-string scaffold
- identical IIFE structure
- identical pattern of appending malicious code **after** a valid config export
- only the `_V` tag differs visibly between the samples

Per-sample variant tags

Repository	File	`_V` tag
Staking-Facilities-GmbH/ReactCosmosDelegationUI	`nwb.config.js`	`8-st21`

Repository	File	_V tag
subramanianv/CertificateVerification	`truffle.js`	`8-st57`
scholtz/covid-sk	`vue.config.js`	`8-st46`
scads-io/frontend	`next.config.js`	`8-778`

Implication

Based on direct inspection, these are not four unrelated obfuscators. They are the same loader family deployed into different framework ecosystems.

Deobfuscation result for the stage-0 payloads

The four samples expose the same stage-0 loader pattern. The first visible deobfuscation routine in the code (`MDy`) is consistent with the loader family described by Ransom-ISAC. Ransom-ISAC says the stage-0 loader:

1. deobfuscates strings using a custom character-shuffling routine,
2. queries TRON for a pointer and uses Aptos as fallback,
3. reverses or extracts data to obtain a BSC transaction hash,
4. calls BSC RPC with `eth_getTransactionByHash`,
5. extracts encrypted data from `transaction.input`,
6. XOR-decrypts the result,
7. executes Payload 1 inline with `eval()`, and
8. retrieves and launches Payload 2 as a detached background process.
   Source: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/

Ransom-ISAC also explicitly documents the two XOR keys and the exact index-chain / payload-chain sequence used by this loader family. Source: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/

Practical interpretation

For the four GitHub samples in scope here, the most accurate statement is:

- **the repo-visible payloads are stage-0 loaders**,
- **they match the publicly documented XCTDH stage-0 family**, and
- **their purpose is to retrieve and execute later payloads from blockchain-hosted data rather than to contain the entire final malware body in the repo itself**.

Known IOCs documented publicly for this loader family

The following values are already documented publicly by Ransom-ISAC as XCTDH stage-0 indicators.

Stage-0 indexing and payload retrieval IOCs

Type	Indicator	Source
TRON address	`TMfKQEd7TJJa5xNZJZ2Lep838vrzrs7mAP`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
TRON address	`TXfxHUet9pJVU1BgVkBAbrES4YUc1nGzcG`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
Aptos address	`0xbe037400670fbf1c32364f762975908dc43eeb38759263e7dfcdabc76380811e`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
Aptos address	`0x3f0e5781d0855fb460661ac63257376db1941b2bb522499e4757ecb3ebd5dce3`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
BSC tx hash	`0xf46c86c886bbf9915f4841a8c27b38c519fe3ce54ba69c98d233d0ffc94d19fc`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
BSC tx hash	`0xd33f78662df123adf2a178628980b605a0026c0d8c4f4e87e43e724cda258fef`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
BSC address	`0x9BC1355344B54DEDf3E44296916eD15653844509`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
XOR key	`2[gWfGj;<:-93Z^C`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/

Type	Indicator	Source
XOR key	`m6:tTh^D)cBz?NM]`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/

Related later-stage / cluster IOCs documented publicly

Type	Indicator	Source
URL	`hxxp://23[.]27[.]20[.]143:27017/$/boot`	https://www.esentire.com/blog/north-korean-apt-malware-analysis-dev-popper-rat-and-omnistealer-everyday-im-shufflin
URL	`hxxp://154[.]91[.]0[.]103:27017/$/boot`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
XOR key	`ThZG+0jfXE6VAGOJ`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
IP	`136[.]0[.]9[.]8`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
IP	`166[.]88[.]4[.]2`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
IP	`23[.]27[.]202[.]27`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
IP	`23[.]27[.]120[.]142`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
IP	`202[.]155[.]8[.]173`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
IP	`198[.]105[.]127[.]210`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
IP	`166[.]88[.]134[.]82`	https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/

Important caveat on domains

The following domains are legitimate public infrastructure and should **not** be treated as globally malicious by themselves:

- `api[.]trongrid[.]io`
- `fullnode[.]mainnet[.]aptoslabs[.]com`
- `bsc-dataseed[.]binance[.]org`
- `bsc-rpc[.]publicnode[.]com`

Ransom-ISAC documents these as part of the retrieval flow, but the services themselves are public infrastructure. TronGrid is documented by TRON as an API service, and Next.js / Vue CLI / Truffle docs likewise confirm that the infected files are executable configuration files, which is why the placement is effective. Sources: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/ and https://developers.tron.network/docs/trongrid

Detection opportunities

High-confidence static repo-hunting patterns

The following strings and patterns are useful for identifying related repositories because they are visible in the infected config files themselves:

- `global['_V']`
- `global['r']=require`
- `function MDy(`
- a long appended one-line blob after a valid `module.exports = ...` statement
- malicious code embedded in one of these project-root config files:
  - `next.config.js`
  - `vue.config.js`
  - `truffle.js` / `truffle-config.js`
  - `nwb.config.js`
  - `tailwind.config.js` (documented by Ransom-ISAC in Part 1)

Ransom-ISAC has also published YARA strings for this family, including `global['_V']`, `global['r']`, `.charAt`, `.substr`, and `require`. Source: https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/

Cross-campaign scale signal and regex hunt seeds

Related open-source tracking from OpenSourceMalware documents a separate but similarly broad config-file injection campaign (PolinRider), reporting infection across hundreds of public repositories. Source: https://github.com/OpenSourceMalware/PolinRider

For practical repo hunting, this is a useful reminder that high-volume campaigns often expose stable string-level markers that can be scanned quickly at scale. In the PolinRider dataset, publicly documented examples include:

- marker string: `rmcej%otb%`
- variable/function artifact: `_$_1e42`
- string pairs used in suggested rule logic: `global['!']` with seed values like `2857687` / `2667686`

- repeated target file families such as `postcss.config.mjs`, `tailwind.config.js`, and `eslint.config.mjs`

These specific markers are campaign-specific and should not be conflated with XCTDH indicators; they are included here as an operational pattern for defenders building broad GitHub hunting pipelines.

Behavioral patterns worth detecting

- toolchain processes unexpectedly making outbound requests to public blockchain APIs or RPC endpoints during config evaluation
- `eth_getTransactionByHash` calls issued by developer tooling rather than blockchain applications
- Node child-process launches from configuration files
- detached `node -e` or equivalent background execution after config loading

Defensive takeaways

1. Treat **JavaScript configuration files** as code, not as harmless metadata. This is explicitly true for Next.js, Vue CLI, Truffle, and `nwb`. Sources: hxxps://nextjs[.]org/docs/pages/api-reference/config/next-config-js, hxxps://cli[.]vuejs[.]org/config/, hxxps://archive[.]trufflesuite[.]com/docs/truffle/reference/configuration/, hxxps://www[.]npmjs[.]com/package/nwb
2. Build or CI systems that only scan application source and ignore root config files will miss this class of intrusion.
3. Repository age or apparently old commit timestamps are weak trust signals when the content is inconsistent with the claimed timeline.
4. Blocking entire public blockchain domains is likely to create false positives; precise detection should focus on exact wallet addresses, transaction hashes, XOR keys, and the stage-0 code pattern.

Limitations

- This article does not claim to have independently executed the live, on-chain stage-1 or stage-2 payloads during preparation.
- The repo-side stage-0 similarity is based on direct static inspection of the four config files listed in scope.
- The later-stage infrastructure, malware names, and broader cluster context are drawn from the cited public reports.

Conclusion

The four repositories analyzed here are best understood as **trojanized public GitHub projects carrying the same stage-0 JavaScript loader family in different framework ecosystems**. The loader is not merely “obfuscated” in a generic sense; it matches a publicly documented XCTDH / DEV#POPPER retrieval chain that uses blockchain-hosted data for pointering, payload retrieval, and resilience.

For defenders, the main lesson is that **root configuration files are executable trust boundaries**. For researchers, the main lesson is that **content-based analysis beats timestamp-based trust** when

repository history appears inconsistent with the embedded infrastructure.

Reference URLs (defanged)

Primary sample URLs

- hxxps://github[.]com/Staking-Facilities-GmbH/ReactCosmosDelegationUI/blob/HEAD/nwb.config.js
- hxxps://github[.]com/Staking-Facilities-GmbH/ReactCosmosDelegationUI/commit/06931046a0086c02b95a86713ce90683a54d08fd
- hxxps://github[.]com/subramanianv/CertificateVerification/blob/HEAD/truffle.js
- hxxps://github[.]com/scholtz/covid-sk/blob/HEAD/vue.config.js
- hxxps://github[.]com/scads-io/frontend/blob/main/next.config.js

External references

- https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist/
- https://ransom-isac.org/blog/cross-chain-txdatahiding-crypto-heist-part-4/
- https://www.esentire.com/blog/north-korean-apt-malware-analysis-dev-popper-rat-and-omnistealer-everyday-im-shufflin