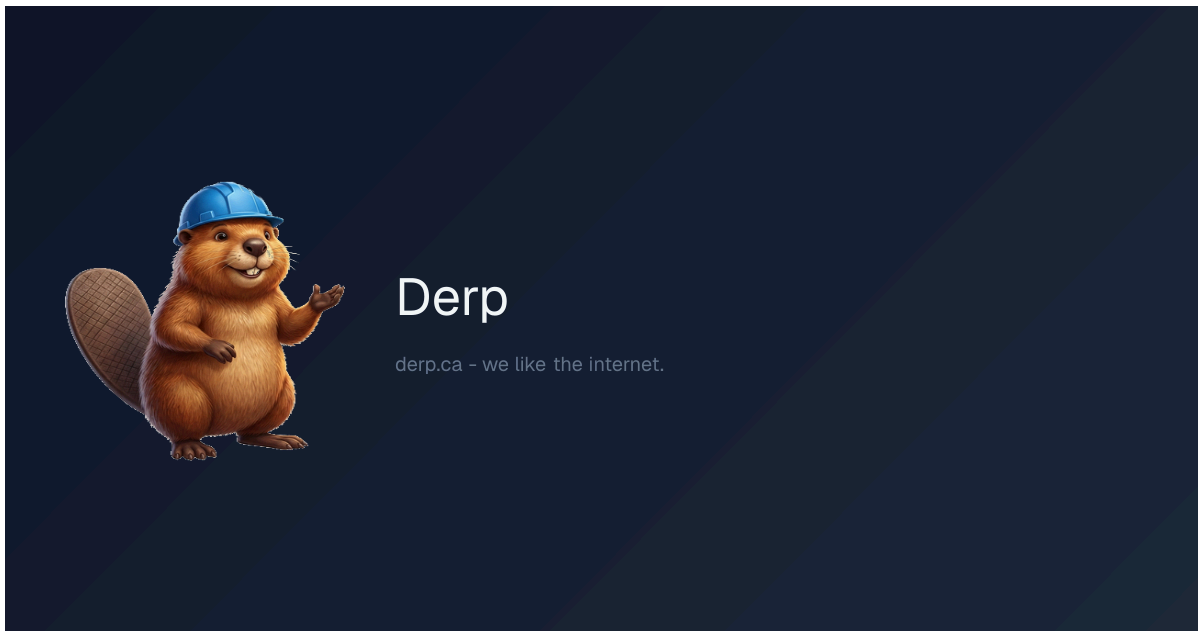


TryNodeUpdate turns GitHub and BSC into a TCP control lane

 derp.ca/research/trynodeupdate-github-node-bsc-contract-c2

Kirk

April 24, 2026



malware github blockchain nodejs windows reverse-engineering

TryNodeUpdate is a Windows malware chain that uses GitHub as its first controller host and a BNB Smart Chain contract as its backend pointer. The original PowerShell sample installs a scheduled task, fetches a Node.js controller from a public GitHub repository, passes a contract address into that controller, and lets the controller resolve a raw TCP endpoint from the chain.

The backend is not in the PowerShell. It is not hardcoded as a normal hostname in the Node controller either. The controller reads `TUNNEL_URL` from a contract call, strips `tcp://`, connects to the returned host on port `8446`, and starts a newline-delimited JSON control lane. On elevated Windows hosts, it downloads a native helper named `rpc.exe` from the same backend. The helper then reconnects as `cpp-tcp-client` and runs its own contract resolver, heartbeat, upload path, and local helper branches.

This sits near earlier Derp blockchain-C2 work, but the shape is different. [OCRFix](#) used BNB Smart Chain contracts to store C2 URLs. [HellsUchecker](#) used EtherHiding to pull encrypted config. TryNodeUpdate uses public GitHub for the first controller, public BSC RPC for the lookup, and raw TCP for the control channel.

If you operate a threat intelligence platform with API access and can provide a researcher account, please reach out to . Additional data sources directly increase the quality and coverage of the threat intel published here.

Sample overview

Artifact	SHA256	Role
4.ps1	b5b771bd9796284faa24f4f9c45d1e77567432049b1e897416bf6f6534b659e9	PowerShell bootstrap
blockchain.js / main.js	5e80a76a758262a1eab6c9d0371e9921f30bc744de55cf01e4c16051be2e8a12	GitHub-hosted Node controller
nodeupdate.vbs	89a6ad4f88bf731b59b690aac4f0082bb25b22309478a7d5d04b702f07b68287	Rendered VBS launcher
client-config.json	897a8df220b0813794d465d2b44adfd5a7b253660b77b75115a905675f0f80bc	Client ID state
rpc.exe	42f469992efb5e80e4ddedf21be8504b5d7e0b4ec78f72511f94170f567bdad5	Native Windows helper
new_write.js	32cce13ce1d029abfd14df74a755f74a88025c497d122b5aaa53a0c35e54024a	Complete JS payload received by the helper

The starting sample was [260422-wfaznads5j](#). The saved sandbox archive preserved [main.js](#), [nodeupdate.vbs](#), [client-config.json](#), and [rpc.exe](#) from the infected working directory, so the Node-to-native handoff is not inferred from strings. The packet capture also reassembles the same [rpc.exe](#) bytes from the backend stream.

Chain

```
4.ps1
-> %LOCALAPPDATA%\TryNodeUpdate\
-> nodeupdate.vbs
-> node-v20.11.0-win-x64\node.exe
-> main.js from raw.githubusercontent.com
-> BSC eth_call
-> JSON.TUNNEL_URL
-> raw TCP backend on 8446
-> rpc.exe
-> cpp-tcp-client
-> uploaded new.js
```

The PowerShell bootstrap checks <http://www.google.com>, creates [%LOCALAPPDATA%\TryNodeUpdate](#), downloads the controller from:

<https://raw.githubusercontent.com/1CodeDev-hub/electronAI/refs/heads/main/blockchain.js>

If the expected Node runtime is missing, it fetches:

<https://nodejs.org/download/release/v20.11.0/node-v20.11.0-win-x64.zip>

Persistence is a scheduled task named `TryNodeUpdateTask`. The task runs:

```
%SystemRoot%\System32\wscript.exe //nologo "%LOCALAPPDATA%\TryNodeUpdate\nodeupdate.vbs"
```

The VBS launcher starts Node against `main.js` with one important argument:

```
BLOCKCHAIN_CONTRACT_ADDRESS=0x75E1eDFA0d0f96D5f8F228358376d6ecdB22d802
```

That value is a contract lookup input. It is not the C2 server.

The GitHub controller

The public GitHub repository `1CodeDev-hub/electronAI` was the source of the controller pulled by the PowerShell sample. Metadata collected on April 24, 2026 showed the repo was created on November 5, 2025, last pushed on April 22, 2026, and had 40 sampled commits.

The commit history lines up with the delivery chain rather than a normal app:

Date	Commit message
2025-12-14	Create install script for Node.js on Windows
2025-12-14	Refactor Node.js installer to use BITS for downloads
2026-01-22	Add Node.js installer script for user-level installation
2026-01-22	Implement scheduled task for node update
2026-03-25	Update main.js download URL and add exit code
2026-03-25	Create VBS script and execute it silently

Date	Commit message
2026-04-22	Add files via upload

The repo also had a release named `For downloads`, tag `danlod`, published on March 24, 2026. At collection time it exposed nine assets, including `rate.cmd`, `rate.js`, `rate.vbs`, `RateConformation.exe`, several VBS wrappers, and `task.exe`. The `task.exe` asset was added on April 23, one day after the selected sample.

That repo context is useful, but it is not enough on its own. The files from this run close the gap: `4.ps1` downloads `blockchain.js`, the sandbox archive preserves the matching `main.js`, and the packet capture shows that controller talking to the contract-derived backend.

Contract-backed backend resolution

The Node controller is obfuscated with a rotated string table. Once decoded, the backend path is direct.

The controller parses `key=value` arguments, reads `BLOCKCHAIN_CONTRACT_ADDRESS`, and sends a BSC `eth_call` using selector:

```
0x6d4ce63c
```

The controller has a list of public BSC RPC endpoints, including:

```
https://bsc.blockrazor.xyz
https://bsc.therpc.io
https://bsc-dataseed2.bnbchain.org
https://bsc-dataseed.bnbchain.org
https://bsc-dataseed-public.bnbchain.org
```

The contract response is ABI-decoded as a string. The decoded string is JSON. The controller reads:

```
TUNNEL_URL
```

On April 22, 2026, three tested contract inputs all returned:

```
{"TUNNEL_URL": "tcp://206.206.127.94:8446"}
```

Those inputs were:

Contract	Source
0x75E1eDFA0d0f96D5f8F228358376d6ecdB22d802	Original PowerShell sample
0xB9F8e457Cd0E97f0CF7aEd57D0654F1Ad61759C1	Repo variant
0x5728Fe164B5a9c599325F6fed9F744306F4D772b	Baked-in fallback

The controller strips `tcp://`, splits the host and port, and opens a raw TCP socket. The protocol is newline-delimited JSON, not HTTP.

Packet capture

The saved packet capture gives the cleanest view of the handoff from the Node controller to the native helper:

Time	Observation
82.792040	DNS answer for <code>bsc.blockrazor.xyz</code>
82.830666	TLS client hello with SNI <code>bsc.blockrazor.xyz</code>
82.946166	TCP SYN to <code>206.206.127.94:8446</code>
83.357425	DNS answer for <code>api.ipify.org</code>
84.528915	Second TCP SYN to <code>206.206.127.94:8446</code>
85.835952	Helper TCP SYN to <code>206.206.127.94:8446</code>

Three backend streams matter:

Stream	Meaning
32	Node <code>register</code> , <code>rpc-download-request</code> , helper chunks, <code>rpc-download-end</code>
34	Node <code>register</code> , <code>rpc-download-meta-request</code> , <code>rpc-download-meta</code>
36	Launched helper registers as <code>cpp-tcp-client</code> and sends periodic heartbeats

The helper metadata returned by the backend names `rpc.exe`, size `886272`, mtime `2026-04-13T13:09:57Z`, and SHA256:

The helper reassembled from stream 32 is byte-identical to the `rpc.exe` preserved in the sandbox archive. That proves the handoff without needing live infrastructure.

What `rpc.exe` adds

`rpc.exe` is not a downloader stub. It repeats the contract-mediated backend lookup in native code and then runs its own control client.

Native analysis mapped the helper into the same broad pieces as the Node controller:

Area	What the helper does
Backend selection	Builds the BSC <code>eth_call</code> , decodes the ABI string, and extracts <code>TUNNEL_URL</code>
Local state	Reads and writes the client config and PID files beside the helper
Registration	Sends host, user, privilege, platform, CPU, memory, public IP, and contract fields
Transport	Connects over raw TCP, receives newline-delimited JSON, and sends JSON responses
Keepalive	Sends 30-second heartbeats
Uploads	Receives files in start, chunk, and end messages

The helper's register object includes host and user data, admin state, platform details, CPU and memory fields, public IP, the contract address, and:

```
connection_platform: cpp-tcp-client
```

The same helper owns the upload path used to write `new.js`:

Message type	Action
<code>file-upload-start</code>	Resolves destination path and opens the active file handle
<code>file-upload-chunk</code>	Base64-decodes inbound data and writes decoded bytes

Message type	Action
<code>file-upload-end</code>	Closes the handle and emits <code>file-upload-complete</code>

In the observed path, chunks are written silently. The helper sends completion only after `file-upload-end`.

Later controlled runs tied the live `new.js` writes back to that native upload chain. Short runs produced exact prefixes of the final payload. A fresh client ID and longer run produced a complete 1518476-byte `new_write.js` payload that passes `node --check`.

The complete `new_write.js` uses the same rotated string-table family as `blockchain.js`. After decoding, it exposed the same main controller command surface as the GitHub controller, with `file-zip` and `file-unzip` added.

Endpoint drift

The original packet capture and April 22 contract read point at:

```
206.206.127.94:8446
```

Later controlled runs recorded the same contract returning:

```
84.200.87.236:8446
```

That distinction matters. `206.206.127.94:8446` is the backend proven by the original packet capture. `84.200.87.236:8446` is proven by later run logs, not by the first capture.

Passive enrichment put `206.206.127.94` in AS396356 `Latitude.sh` with PTR `host.trynode.cloud`. Current forward and reverse pivots tied the host to `www.trynode.cloud`, `mesh.trynode.cloud`, `trynode.info`, and `mail.trynode.info`. The same IP exposed a mixed service surface: SSH, Apache redirects to `https://grailed.lnks.space/`, PPTP, Express, and a `phpMyAdmin` login page on `8232`.

The later observed endpoint `84.200.87.236` landed in AS214036 `Ultrahost, Inc.` in Germany. Passive data tied it to `sjjh.jacktake.website`, `80fa.jacktake.website`, `pmta236.qigaf84.caymedin.com`, and `doodles.yachts`. It also exposed a `phpMyAdmin` login page on `8232`.

Those adjacent surfaces are pivots, not automatic malware lanes. The proven malware lane is raw TCP on `8446`.

Boundaries

There are a few things this case does not prove.

`bsc.blockrazor.xyz` is treated here as a public BSC RPC dependency. The evidence shows the malware used it to read a contract. It does not show BlockRazor operating the malware backend.

85.203.46.39 appears as the helper-reported public_ip in the detonation register. It is not the backend.

Cloudflare IPs for bsc.blockrazor.xyz are Cloudflare edges. They are not operator-owned infrastructure.

The native helper has branches for embedder.exe, get_data.exe, and output.zip, but those side-helper binaries were absent from the saved sandbox archive. The branches are real. The helper contents are not recovered here.

The backend-side rule that decides whether a session gets a prefix of new.js or the complete new_write.js remains server-side. Local evidence rules out a one-chunk client cap: every shorter artifact is an exact prefix of the complete payload, and longer runs with fresh IDs received more data.

Indicators

Type	Value
Scheduled task	TryNodeUpdateTask
Working directory	%LOCALAPPDATA%\TryNodeUpdate\
Controller URL	https://raw.githubusercontent.com/1CodeDev-hub/electronAI/refs/heads/main/blockchain.js
Node runtime URL	https://nodejs.org/download/release/v20.11.0/node-v20.11.0-win-x64.zip
Contract	0x75E1eDFA0d0f96D5f8F228358376d6ecdB22d802
BSC selector	0x6d4ce63c
Original backend	206.206.127.94:8446
Later observed backend	84.200.87.236:8446
RPC endpoint dependency	https://bsc.blockrazor.xyz
Public IP check	https://api.ipify.org/?format=json

The useful detection shape is the combination: a user-level TryNodeUpdate directory, Node runtime bootstrap, raw.githubusercontent.com controller fetch, BSC eth_call traffic, and raw TCP on 8446 to the contract-returned host. Any one piece can look normal. Together, they describe the chain.

Kirk

I like the internet. Want to get in touch?