

# 2026 eCash Node Security Audit Report

Conducted by: teamsocket.net

## Version 1.2

| Date       | Version     | Changelog                             |
|------------|-------------|---------------------------------------|
| 2026-09-28 | Version 1.2 | Changed title                         |
| 2026-09-26 | Version 1.1 | Clarified some sentences + formatting |
| 2026-09-08 | Version 1.0 | Initial Report                        |

## Executive Summary

This report details the results of a security evaluation of the eCash node software built by Bitcoin ABC conducted from May to September 2026.

Vulnerabilities discovered and responsibly disclosed to Bitcoin ABC under this audit, by severity:

| Critical | High | Medium | Low |
|----------|------|--------|-----|
| 5        | 3    | 5      | 10  |

And some key metrics related to those responsible disclosures:

| Avg Time to Respond | Avg Time to Fix | Avg Time to Deploy |
|---------------------|-----------------|--------------------|
| 99 minutes          | 2.3 days        | 5 days             |

All of the described vulnerabilities have been fixed and published in a public release of the Bitcoin ABC software unless otherwise noted. Bitcoin ABC was able to demonstrate the full capabilities of their team by consistently and rapidly releasing fixes as these vulnerabilities were discovered.

## Projects Other Than Bitcoin ABC

Although not originally in scope for this audit, vulnerabilities discovered and separately responsibly disclosed to projects other than Bitcoin ABC occurred frequently enough that they deserve a tally here:

| Critical | High | Medium | Low | Informative* |
|----------|------|--------|-----|--------------|
| 0        | 1    | 2      | 0   | 3            |

(\*) Informative means the disclosure was made but the receiving team decided to class it as not a vulnerability and that the report was informative only.

## Scope

Source Code Repository: <https://github.com/Bitcoin-ABC/bitcoin-abc>

Code Under Review: All files under `src/` and their associated functional tests under `test/functional`. Also within scope is the Chronik indexer source code under `chronik/`. While Chronik is an optional module of the node software, users of Chronik require its reliability to be as good as the node's RPC so that it can be used in its place. Therefore, it is treated as a critical component of the node software. Note that it is not standard industry practice for a node's indexer to be treated as integral to the node. This appears unique to Bitcoin ABC.

For the Code Under Review, the process of applying changes to that code will also be examined.

## Objective

Identification, responsible disclosure, cooperation in fixing, and coordination in deploying fixes of potentially exploitable vulnerabilities in the eCash node software. This process was conducted repeatedly until no further medium or higher severity findings could be identified.

Recommendations and fixes outside of scope (documentation, libraries, etc.) have been provided on a best effort basis when relevant to the outcomes of this audit.

## Context

In May 2026, I (Roqqit of teamssocket.net) began conducting vulnerability scans to test the efficacy of LLM agent harnesses on the Bitcoin ABC code base. After multiple findings, the Bitcoin ABC team requested a more comprehensive security audit of the eCash node software.

I have prior experience developing on the Bitcoin ABC code base with over 161 commits at the time of this writing.

## Code Overview

The Bitcoin ABC repository is a monorepo with the eCash node software implemented in C++ and organized under `src/`. The unit tests for the node software primarily reside under `src/test/` with some modules having their own dedicated unit tests (ex: `src/avalanche/test/`). The functional test suite, implemented in Python 3, is located separately from the node source under `test/functional/`.

## Development Practices

Bitcoin ABC's development philosophy is documented in [CONTRIBUTING.md](#) at the top level of the repository. Generally, that philosophy encourages small, focused changes that are reviewed quickly and made available to all developers on the master branch with git used for source control. Code review is conducted using a Phabricator instance at [reviews.bitcoinabc.org](https://reviews.bitcoinabc.org). The patching process is rebase-only on the master branch, meaning changes are never merged. The patch is kept up-to-date during the review process instead of brought up-to-date during the acceptance of the patch, which Bitcoin ABC refers to as "landing" a patch. Despite being rebase-only on master, Bitcoin ABC does encourage the use of branches for local development.

Bitcoin ABC has little tooling guidance for authoring code changes. Preferences are largely left up to the author.

When a patch is ready for peer review, Bitcoin ABC uses the [arcanist tool](#), with the author running `arc diff` to begin the submission. At this stage, the author becomes prompted for code linting, if necessary. The author must also supply a description of the change along with testing instructions.

Shortly after submitting a patch for review, Bitcoin ABC's [TeamCity continuous integration \(CI\) system](#) automatically begins running pre-selected tests on the patch. If tests pass, developers from the Bitcoin ABC team begin reviewing for impact, intent, code quality, development philosophy (small, focused changes), test coverage, etc. Any Bitcoin ABC team member may review any patch. There are no fine-grained Access Control Lists (ACLs) for the review process, however most reviewers focus on their areas of expertise. Bitcoin ABC has informed me that only active and/or vetted developers are granted membership for code review. Two-factor Authentication (2FA) is enforced for all team members with admin privileges on Phabricator.

If the reviewers deem changes to be required, the author must modify their patch and then update their submission with `arc diff`. Upon every resubmission, the CI system re-runs the tests. This process repeats until all tests pass and all reviewers have accepted the submission.

When a patch has been accepted, the author must trigger landing the change with `arc land`. The patch becomes available on the master branch immediately and the CI system conducts another round of tests to ensure no regressions are present. This round of tests is significantly more comprehensive than during the review stage.

If any test failures are found during the post-submission period, the Bitcoin ABC team is notified of the regression. After a brief investigation, the team will decide either to revert the offending patch or fix-forward, with fix-forward being preferred unless the regression cannot be fixed in a reasonable timeframe. In either case, a new patch is submitted and undergoes its own review process. This process is repeated until all tests pass on the master branch, known as master being "green".

Bitcoin ABC has also indicated the existence of a non-public infrastructure repository for maintaining sensitive services and configurations that are critical to the development process. The servers utilizing this infrastructure were configured to: disable remote root login, root password was removed, and access limited to whitelisted public keys. By default, only one person had access to all keys and policy is to grant access to specific machines to other team members only on an as-needed basis. Key holder succession plan in the event of unplanned discontinuance is to rebuild infrastructure from the repository with new keys.

### **Commentary on Development Practices**

I observed these development practices first hand during this audit when I submitted some of the fixes described in the Vulnerabilities section.

Bitcoin ABC's development practices are a blend of traditional open source software practices and those of high-scale software businesses. While Github dominates many open source projects' development style (encouraging merging), Bitcoin ABC has explicitly chosen not to adopt this development style in favor of well-vetted practices developed at companies like Google and Meta. This allows Bitcoin ABC to maintain a rapid pace of development, even with a small team.

The lack of fine-grained ACLs remains a place for the Bitcoin ABC team to explore as the team grows larger. It does not negatively impact development capabilities so long as Bitcoin ABC is able to maintain its high-ownership culture. Note that the culture of the team is not explicitly documented anywhere. I am told the expectations are quite simple: high competence contributors who are respectful will be treated likewise. Many open source projects have a cultural document such as a Code of Conduct, but Bitcoin ABC has explicitly determined not to codify one, as they believe it distracts from their mission and encourages political conflict.

I noted the existence of the non-public infrastructure repository. This is standard practice in open source critical infrastructure projects.

The security protocols protecting the development infrastructure appear sound. I do note only having one person with direct access to infrastructure keys is not ideal, but being able to quickly replace infrastructure from the private repository is a reasonable mitigation. I also note the authentication systems were not homogeneous (ie a separate auth for Phabricator compared to TeamCity). This does not make the system inherently insecure, but it does incur a small operational cost if Bitcoin ABC wishes to keep these systems in sync.

### **Release Process**

Bitcoin ABC's release process generally goes as follows, on a cadence of every two weeks:

1. Select a commit hash from master (usually the most recent commit).
2. Tag the commit in git.
3. Generate release binaries from that commit.
4. Cryptographically sign the release binaries.
5. Upload binaries and signatures to [download.bitcoinabc.org](https://download.bitcoinabc.org)
6. Mirror those binaries and signatures to [Github](https://github.com)
7. [bitcoinabc.org/releases](https://bitcoinabc.org/releases) becomes automatically updated.

Bitcoin ABC also has a flag-day upgrade release schedule that enables major features to activate in a coordinated fashion. The flag-day upgrade occurs every May 15 and November 15, with the next major version being released shortly after the upgrade. Node operators must be running the most recent major version during and after the flag-day upgrade. This results in a major version release cadence of roughly every 6 months.

Bitcoin ABC also conducts hotfix releases on an as-needed basis (usually high or critical severity vulnerability fixes).

Although I did not participate in the release process, it is apparent from the many releases I observed during this audit that the process is at least consistent with my documentation here.

### **Commentary on Release Practices**

Looking at the past 180 days, Bitcoin ABC averages 2+ commits per day on the node software which sets an expectation of roughly 30 changes per release. This cadence (and the flag-day upgrade cadence) is fairly typical of similar projects. That is, projects that have high risk during major feature releases do so less frequently while still maintaining patch releases on a consistent basis.

## Secure Disclosure Policy & Commentary

At the start of this audit, Bitcoin ABC's security policy was located in a file called DISCLOSURE\_POLICY.md. I noted during this audit that SECURITY.md was the prevailing naming standard and that Github was failing to present it under Bitcoin ABC's Security tab. I submitted a fix (D20452) so that the policy now sits at SECURITY.md.

While my disclosures were submitted more securely to Bitcoin ABC than via email (and so did not need to be separately encrypted), I did verify that Bitcoin ABC's security disclosure GPG key was both accessible and valid.

I reviewed Bitcoin ABC's disclosure policy and found it largely in line with industry standards.

I also reviewed discoverability of the security policy and found that the policy was not served at

`/.well-known/security.txt`

from e.cash nor bitcoinabc.org. I recommend Bitcoin ABC serve the security policy at these URLs since it is emerging as standard practice in the industry.

## Security Evaluation

The first scan in this report began on commit `1c4b1ca42f38da8b366d62451481bd003d89b547` (May 22, 2026) and the final scan was conducted on commit `0a704c5a875a9bfd2daf701ec6df5fcd60a04e4f` (September 2, 2026).

The setup for discovery of vulnerabilities was primarily composed of:

- A specialized Docker container for sandboxing
- opencode LLM harness
- Specialized LLM skills and prompt formats designed to pinpoint vulnerabilities in bitcoin-like software
- Various AI models for discovery, notably Grok 4.3 - 4.5 and Kimi K3.

## Methodology

The vulnerability scans generally followed this methodology:

- Constrain an LLM to a specific area of code and/or type of exploit and request it to find vulnerabilities.
- Challenged LLM against its findings via Q&A and proof-of-concept test coverage.
- Summarization of findings if they passed the challenge step.
- For medium and below severity issues, the findings were passed immediately to the Bitcoin ABC team.
- For high and critical severity issues, further proof-of-concept testing was conducted manually and/or a patch was authored manually to fix the issue.

All findings and patches from the above process were shared privately for the Bitcoin ABC team to consider. We then promptly cooperated to determine appropriate timelines for applying the fix, conducting a release (hotfixing if necessary), and gradual public disclosure.

## Vulnerabilities

Short names of vulnerabilities provided for convenience in parentheses. All versions of in-consensus (roughly the last 6 months worth of releases) Bitcoin ABC nodes were vulnerable at the time of discovery unless otherwise noted.

Some vulnerabilities impact other projects in addition to Bitcoin ABC. Those notes can also be found in the details of each vulnerability below.

### [Critical] REDACTED

This disclosure is ongoing and has been redacted until all-clear for publication.

### [Critical] REDACTED

This disclosure is ongoing and has been redacted until all-clear for publication.

**[Critical] REDACTED**

This disclosure is ongoing and has been redacted until all-clear for publication.

**[Critical] REDACTED**

This disclosure is ongoing and has been redacted until all-clear for publication.

**[Critical] REDACTED**

This disclosure is ongoing and has been redacted until all-clear for publication.

**Avalanche Slot Compaction Bug (ASCB)**

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| High     | Denial of Service (DoS) | Low                   | Medium                |

A bug in Avalanche slot compaction could be abused such that nodes become unable to select valid peers for polling.

**Impact**

In the worst case scenario, an attacker trigger DoS network-wide by bloating the Avalanche peermanager slots to the point where all nodes would fail to poll for all Avalanche votes, effectively disabling Avalanche.

An attacker requires a valid proof to execute the attack. The more an attacker spends on their proof UTXOs (minimum ~\$700 per proof as of the time of discovery of this vulnerability), the the easier and quicker the attack is to pull off successfully. An attacker with 2 or more proofs (minimum ~\$1,400) can invoke the worse case DoS described above.

**Details**

The bug was introduced during the initial development of Avalanche in [D8610](#). With the introduction of `newslots` in `compact()`, the line `prevStop = slots[i].getStop();` appears to have been missed when it should have been refactored to accommodate it.

I submitted patch [D20385](#) to fix the issue and included additional test coverage to prevent regressions.

**Invalidation Coin Cache Miss (ICCM)**

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| High     | Denial of Service (DoS) | High                  | Low                   |

A coin cache miss on the Avalanche tx-invalidation path could leave the `TransactionInvalidated` validation callback with unusable spent-coin data. If a transaction is invalidated in the same vote round as its parent, the parent's recursive mempool removal also drops the child's mempool coins. A later `GetSpentCoins` for the child then misses, so the callback no longer carries the inputs Chronik needs to reconstruct the tx.

**Impact**

On Chronik-enabled nodes, reconstruction failure in `handle_tx_invalidated` is fatal (`ok_or_abort`), aborting the process.

After [D20091](#), the same miss could also crash outside Chronik via unsafe use of an empty `std::optional<Coin>` inside `GetSpentCoins`.

Any node running Avalanche transaction preconsensus is in scope; Chronik indexers and other spent-coin consumers of `TransactionInvalidated` are the sharp edge. Coordinated invalidation of parent/child packages can disrupt those nodes.

**Details**

Spent coins were attached to the invalidation callback in [D18438](#) so Chronik could recover token/plugin metadata after mempool removal.

GetSpentCoins assumed every input coin was present (Assume + push). That assumption fails when:

1. Parent P and child C (spending P's mempool output) are both Avalanche-invalidated in one response/round.
2. P is processed first: removeRecursive removes P and its descendants, including C's mempool coins.
3. C is processed next: GetSpentCoins runs against a view that no longer has C's inputs.
4. Pre-D20091 this yielded default/empty Coin entries (garbage callback data). D20091 made the miss worse: GetSpentCoins still did Assume(coin.has\_value()) then \*coin, which is invalid when the optional is empty.

Chronik then fails bridging/reconstructing the tx from that spent-coin vector and aborts.

The vulnerability was fixed in [D20102](#) which:

- Makes GetSpentCoins return std::nullopt on any missing coin.
- Gathers spent coins **before** recursive mempool removal when possible.
- Passes a null shared\_ptr when coins are unavailable, with Chronik (and other defensive checks) skipping instead of dereferencing garbage.

## Sticky Remote Proof (SRP)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| High     | Denial of Service (DoS) | High                  | High                  |

A sticky remote-proof present: true could keep a dangling Avalanche proof eligible as a stake contender after peers had stopped advertising it. Miners and voters would treat that proof as a live contender and could finalize it as the staking-reward winner, while block policy still rejected payouts to that proof, so the mined block was parked/rejected.

### Impact

Affected nodes running staking-reward preconsensus could:

- Poll and vote for a dangling proof that is no longer actually shared by peers.
- Select that proof as the staking-reward winner.
- Mine (or accept as policy-valid locally) a block paying that winner, then see the network park/reject the block.

Result is wasted miner work and disrupted staking-reward agreement until the sticky remote state aged out or was overwritten.

### Details

Remote proofs were append-mostly: statuses for a peer accumulated and were only trimmed by size limits, not replaced per compact avaproofs snapshot.

Compact avaproofs handling only matched shortids against **connected peer** proofs (forEachPeer). Dangling proofs were the intended pullback target of remote presence, but they were not matched. Once peers stopped sending a proof that the local node had moved to the dangling pool:

1. The old remote row stayed present: true (sticky).
2. Later compact messages that omitted the proof did not flip it to absent, because matching never considered the dangling pool.
3. Contender promotion kept dangling proofs alive when isRemotelyPresentProof(proofid) && isDangling(proofid).
4. The miner polled that contender; stakers could vote it in; payout to it still failed staking-reward policy / network acceptance.

The vulnerability was fixed in [D20474](#) which:

- Builds a full remote-proof snapshot by matching compact shortids against both peer proofs and the dangling pool.
- Clears that node's remote proofs and reapplies the snapshot each round, so omitted proofs become absent and stale sticky rows are dropped.

## [Medium] REDACTED

This disclosure is ongoing and has been redacted until all-clear for publication.

## OOB in cashaddr::Decode()

| Severity | Class               | Difficulty to Trigger | Difficulty to Exploit |
|----------|---------------------|-----------------------|-----------------------|
| Medium   | Out of Bounds (OOB) | Trivial               | Unknown               |

Exerpts from the [Full Disclosure Details](#) are provided below.

### Impact

Trivially triggerable out-of-bounds (OOB) in cashaddr::Decode() may trigger a crash. The node software acts as a reference implementation for various libraries and other exogenous code bases, so impact to the end user will depend on how this issue manifests in those softwares.

### Affected Software

| BLOCKCHAIN                       | SOFTWARE                                         |
|----------------------------------|--------------------------------------------------|
| eCash (XEC)                      | <a href="#">Bitcoin ABC Node</a>                 |
| eCash (XEC)                      | <a href="#">Bitcoin ABC CKPool</a>               |
| eCash (XEC)                      | <a href="#">Bitcoin ABC Trezor Firmware</a>      |
| eCash (XEC) + Bitcoin Cash (BCH) | <a href="#">Trezor Firmware</a>                  |
| eCash (XEC)                      | <a href="#">Chronik bitcoinsuite-core Module</a> |
| Bitcoin Cash (BCH)               | <a href="#">Bitcoin Cash Node</a>                |

### Details

At a high-level, cashaddr::Decode() does not appropriately validate the checksum length to be 8 bytes.

It is possible to generate an address with a checksum shorter than 8 bytes that passes initial checksum checks, that later triggers an out-of-bounds, resulting in a `std::length_error: cannot create std::vector larger than max_size()` in `cashaddr.cpp:289` which reads:

```
return {std::move(prefix), data(values.begin(), values.end() - 8)};
```

Bitcoin ABC fixed this vulnerability in: \* Node: [D20444](#) \* CKPool: [PR12](#) \* Trezor Firmware: [PR7750](#) \* Chronik bitcoinsuite-core: [D20548](#)

## Electrum Unbounded Read Buffer (EURB)

| Severity | Class               | Difficulty to Trigger | Difficulty to Exploit |
|----------|---------------------|-----------------------|-----------------------|
| Medium   | Out of Memory (OOM) | Low                   | Low                   |

Certain inputs can trigger unbounded per-connection memory growth in `electrum_codec` which has maximum buffer size. This is fixed in [D20578](#).

This vulnerability can be trivially mitigated with a reverse proxy.

## Signed Overflow in Amount deserialization of decoderawtransaction

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Medium   | Undefined Behavior (UB) | Trivial               | High                  |

Backporting Bitcoin Core [PR20406](#) resolves a signed overflow in Amount deserialization, reachable via `decoderawtransaction`. This is treated as Medium severity due to a history of `decoderawtransaction` being exposed to user input in various services. Bitcoin ABC fixed this in [D20441](#).

### [Medium] REDACTED

This disclosure is ongoing and has been redacted until all-clear for publication.

## Chronik Non-integer EstimateFee Crash (CNEC)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Denial of Service (DoS) | Trivial               | Trivial               |

Non-default configurations expose nodes to a vulnerability in `blockchain.estimatefee` where non-integer input triggers a panic in the Electrum RPC handler. This is fixed in [D20569](#).

## Net Magic/Msg Start Grinding (PR19107)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Denial of Service (DoS) | Low                   | High                  |

A Low severity DoS exists due to Msg Start and Net Magic not being checked until the entire message is received. Bitcoin Core [PR19107] resolves this issue and Bitcoin ABC backported it in [D20045](#) to fix the issue in the ABC code base.

## Signed Overflow in Avalanche Proof Amount Deserialization (UBPROOF)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Undefined Behavior (UB) | Low                   | High/Unknown          |

A signed overflow exists in Avalanche proof Amount deserialization. Fortunately, I can assign this as Low severity because despite being Undefined Behavior, x86 wraps the value and there is no apparent way to cause a crash. The issue was treated with urgency out of caution, and Bitcoin ABC fixed it in [D20397](#).

## Addrman Inbound Stuffer (PR30568)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Undefined Behavior (UB) | High                  | High                  |

A determined attacker can theoretically trigger an overflow in `src/addrman.cpp:842` which reads: `int nId = vvNew[nUBucket][(nUBucketPos + i) % ADDRMAN_BUCKET_SIZE];` That `int` type should have been backported as `nid_type` but was missed in [D18015](#). I resolved this in [D20403](#). In order to execute an attack, an attacker would need to feed new addresses to AddrMan at the maximal rate allowed by the node. Even if the attacker was able to connect to victims' nodes with all inbound connections, it would still take approximately 8 months to trigger the overflow.

## Signed Overflow in prioritizetransaction (PR23418)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Undefined Behavior (UB) | Trivial               | High                  |

A signed overflow exists in `prioritizetransaction`. Bitcoin Core [PR23418](#) resolves this and it was backported into Bitcoin ABC in [D20440](#).

## Rewind() Assertion in Reindexing (PR32878)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Denial of Service (DoS) | Medium                | High                  |

An attacker can theoretically induce an assertion in `Rewind()` while a node is reindexing. Bitcoin Core [PR32878](#) resolves this and it was backported into Bitcoin ABC in [D20436](#).

## GETAVAADDR Use-after-free

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Undefined Behavior (UB) | High                  | High                  |

pnode is dereferenced outside of the `m_connman.ForEachNode()` loop. If the peer is cleaned up via `DisconnectNodes()`, then that dereference is a use-after-free which is undefined behavior. This vulnerability is made harder to exploit by rate limits imposed on `GETAVAAADDR` requests. This behavior was fixed in [D20531](#).

### Chronik Fake Confirmations After Reorg (CFCAR)

| Severity | Class           | Difficulty to Trigger | Difficulty to Exploit |
|----------|-----------------|-----------------------|-----------------------|
| Low      | Data Corruption | High                  | High                  |

A node run with `-chronik=0` during a reorg that is then restarted with Chronik enabled after the reorg can fail to disconnect the stale fork's chain tip. This results in Chronik serving up blocks and transactions as finalized when they are not. This is fixed in [D20577](#).

### VoteRecord inflight underflow

| Severity | Class                                             | Difficulty to Trigger | Difficulty to Exploit |
|----------|---------------------------------------------------|-----------------------|-----------------------|
| Low      | Undefined Behavior (UB) + Denial of Service (DoS) | High                  | High                  |

Theoretically it is possible for a polled item to go stale (removing it from inflight) and then immediately become valid for polling again. This results in the processor holding the item for inflight accounting but the voterecord's inflight counter has underflowed, making it unpollable. Effectively, this becomes a DoS on that polled item. In practice, there does not appear to be a way to weaponize this into an exploit of consequence. This was fixed in [D20534](#).

### Coinbase Token Tx Chronik Plugin Crash (CTTCP)

| Severity | Class                   | Difficulty to Trigger | Difficulty to Exploit |
|----------|-------------------------|-----------------------|-----------------------|
| Low      | Denial of Service (DoS) | Very High             | Low                   |

`BatchProcessor::tx_token_inputs` special-cases coinbase txs and returns an empty `spent_tokens` vec (`Ok(vec[])`), but the coinbase still has 1 input. When a plugin's `lokad` ID matches the coinbase's `OP_RETURN`, `PluginContext::run_on_tx -> bridge_tx` iterates the tx inputs and indexes `tokens[0]` on the empty vector triggering an OOB panic. On restart, resyncing reprocesses the same block resulting in a persistent crash until plugins are disabled. This was fixed in [D20576](#).