BREWTFORCE

Caffeinated node Internal Review

Summary

Project Name	Espresso
Language	Go
Codebase	https://github.com/EspressoSystems/nitro-espresso-integration
Delivery Date	31/03/2025
Team	0xKato, Jarred Parr
Commit(s)	378fd063e4dc5ddf0089410732c73dc205b6d2d9

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
High	1	0	0	0	1	0
Medium	1	0	0	0	0	1
Low	3	0	0	2	0	0

Global-1 | Force inclusion can lead to caff node halt

Category	Severity	Location	Status
Liveness Violation	High	Global	Confirmed

Description

A liveness fault in the batcher can cause the Caff node to stop functioning. The Caff node cannot apply any valid next batch that matches both the L1 inbox state and the expected state transition. This results in loss of liveness but not incorrectness. The issue is not mitigated by TEE security and relies on permissionless batching under a one-honest-node assumption to be avoided.

Delayed Transaction Submitted

A user submits a delayed transaction `Tx_D` directly to the L1 delayed inbox.
`Tx_D` enters the L1 delayed inbox, but is not yet included in any sequencer batch.

Batcher Submits a Batch to Espresso Only

The batcher submits a batch `[Tx_1, Tx_2]` to Espresso.
This batch does not include `Tx_D`

Espresso Finalizes the Batch

Espresso finalizes block #`500` containing `[Tx_1, Tx_2]`.
The Caff node processes this finalized block and applies the state transition:

```
state_root = S1
```

At this point, the Caff node is unaware of `Tx_D`, since it only sees what was finalized by Espresso.

Timeout and Force Inclusion Triggered

After 24 hours, `Tx_D` has still not been included in any batch.
A user calls `forceInclusion()` on L1, triggering the forced inclusion of `Tx_D` into the canonical inbox.
Now, the canonical inbox on L1 looks like:

```
[Tx_D, Tx_1, Tx_2]
```

Inbox Mismatch Detected

The Nitro client (or any canonical L2 verifier) replays the inbox from L1 and sees:

```
[Tx_D, Tx_1, Tx_2]
```

But the Caff node had already applied state transitions for just `[Tx_1, Tx_2]` (no `Tx_D`), and reached `state_root = S1`.

Caff Node is Stuck

Current behavior:
Caff node would stop functioning because there's no valid next batch to apply that matches both the L1 inbox state (which includes `Tx_D`) and the expected state transition based on what was finalized in Espresso.

Recommendation

Since a liveness fault is required, this can be mitigated by a one-honest-node assumption, which is planned for the permissionless batcher upgrade.

Resolution

Will be fixed as part of the permissionless batcher upgrade.

EUT-1 | Spam attack with signed empty blocks

Category	Severity	Location	Status
DoS	Medium	espresso_streamer.go	Confirmed

Description

The caff node uses attestations to validate that a batch comes from the correct batch poster. However, an issue arises with empty batches. Since there are no message indices to enforce progression, the node only verifies the signature. This means any empty block signed by the batch poster can be replayed repeatedly, forcing the caff node to exhaust more resources than necessary.

Recommendation

Instead of returning an empty array of messages if `currentPos == len(payload)` return an error

Resolution

Recommendation will be implemented.

BP-1 | Unsafe order of operations

Category	Severity	Location	Status
Implementation Error	Low	block_processor.go:L408-L414	Acknowledged

Description

Performs a subtraction before checking whether `txGasUsed < dataGas`, which means it may underflow when `dataGas` exceeds `txGasUsed`. Since both are `uint64`, this results in a very large wrapped-around value.

Although `computeUsed` is later overwritten if the underflow condition is detected, the unsafe subtraction still occurs first, which is risky and error-prone.

```
computeUsed := txGasUsed - dataGas
if txGasUsed < dataGas {
        log.Error("ApplyTransaction() used less gas than it should have", "delta", dataGas-txGasUsed)
        computeUsed = params.TxGas      }
else if computeUsed < params.TxGas {
        computeUsed = params.TxGas }
```

Recommendation

Check the condition before subtracting.

Resolution

Acknowledged

BP-2 | Superfluous operation

Category	Severity	Location	Status
Implementation Error	Low	block_processor.go:L271-L273	Acknowledged

Description

The function `extraPreTxFilter(...)` is intended to perform additional pre-transaction validation.

```
if err = extraPreTxFilter(chainConfig, header, statedb, state, tx, options, sender, l1Info);
        err != nil { return nil, nil, err }

func extraPreTxFilter( chainConfig *params.ChainConfig, currentBlockHeader *types.Header, statedb *state.StateDB, state
*arbosState.ArbosState, tx *types.Transaction, options *arbitrum_types.ConditionalOptions, sender common.Address,    l1Info *L1Info, ) error {
        // TODO: implement additional pre-transaction checks
        return nil }
```

it effectively does nothing, all transactions will pass this check unconditionally. This creates a false sense of security and may lead developers to assume extra validation is in place when it isn't.

Recommendation

Either implement the intended logic or remove the call entirely to avoid misleading assumptions in security-critical code.

Resolution

Acknowledged

Global-2 | Incorrect terminology

Category	Severity	Location	Status
Documentation	Low	Global-2	Confirmed

Description

Outdated Naming: `L1Info`

The variable and structure named `L1Info` is misleading in the current context.
It no longer reflects Layer 1 data, instead, it represents data from Espresso messages.

Recommendation

Consider updating the variables to clearly reflex the code

Resolution

Disclaimer

This report is an internal review and should not be considered an “endorsement” or “disapproval” of any particular part of the codebase. It does not provide any warranty or guarantee regarding the absolute bug-free nature of the analyzed technology, nor does it reflect the economics, value, business model, or legal compliance.

This report should not be used to make investment or involvement decisions. It is not a substitute for external reviews and should not be taken as investment advice. Instead, it serves as part of an internal assessment process aimed at helping improve the quality of the code.

The goal is to help reduce attack vectors and risks associated with evolving technologies, we do not claim any guarantees regarding security or functionality of the technology analyzed. We do not guarantee the explicit security of the audited code, regardless of the findings.