BREWTFORCE

Hotstuff-2 Internal Review

Summary

Project Name	Espresso
Language	Rust
Codebase	https://github.com/EspressoSystems/hotshot
Delivery Date	06/03/2025
Team	0xKato, Jarred Parr
Commit(s)	cfcfca617ca769cb8ab2b29a4a8ca0ebf75d4800
Scope	Functions: decide_from_proposal_2, handle_timeout, wait_for_highest_qc

Vulnerability Summary

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

DFP-1 | Liveness failure due to only passing the last element of array

Category	Severity	Location	Status
Liveness Violation	Critical	crates/task-impls/src/helpers.rs	Confirmed

Description

The functions `decide_from_proposal_2` and `decide_from_proposal` iterate in reverse over the decided proposals. They start with the most recent view (view *N*) and step backward by a specified value *x* (i.e., from view *N* to view *N-x*) until the oldest decided proposal is reached.

Below is an example snippet illustrating this reverse iteration:

```
while current_leaf_info.as_ref().is_some_and(|info| info.leaf.view_number() > old_anchor_view){
```

During the execution of this loop, the view information is appended to an array using:
res.leaf_views.push(info.clone());

The final element of the `leaf_views` array will be passed into the `decide_epoch_root` function:

```
if let Some(decided_leaf_info) = res.leaf_views.last() {
        decide_epoch_root(&decided_leaf_info.leaf, epoch_height, membership).await;
```

Within `decide_epoch_root`, a check is performed to verify that the provided `decided_leaf_info` corresponds to the third-to-last block in the epoch. This check is implemented by the `is_epoch_root` function:

```
pub fn is_epoch_root(block_number: u64, epoch_height: u64) -> bool {
        if block_number == 0 || epoch_height == 0 { false }
        else { (block_number + 2) % epoch_height == 0 } }
```

If this edge case is hit, it leads to a liveness failure in the consensus mechanism, preventing progress into the new epoch.

Recommendation

Iterate over all elements in the `leaf_views` array and check each one against the condition `(block_number + 2) % epoch_height == 0`. Synchronize based on the first view that satisfies this condition to avoid a consensus liveness failure.

Resolution

Fixed in 2728

DFP-2 | Superfluous check

Category	Severity	Location	Status
Superfluous code	Low	crates/task-impls/src/helpers.rs	Confirmed

Description

When calling decide_epoch_root, it executes a check we first check if `epoch_height != 0` and then we check `is_epoch_root`

```
if *epoch_height* != 0 && is_epoch_root(decided_block_number, *epoch_height*) {
```

is_epoch_root checks whether `block_number == 0` OR `epoch_height == 0`

```
pub fn is_epoch_root(block_number: u64, epoch_height: u64) -> bool {
        if block_number == 0 || epoch_height == 0 { false }
        else { (block_number + 2) % epoch_height == 0 } }
```

as we can see the initial epoch height check can be removed since its checked in `is_epoch_root`.

Recommendation

Remove the *epoch_height* != 0 check.

Resolution

Acknowledged

Global-1 | Enhance Code Comments for Better Readability

Category	Severity	Location	Status
Documentation	Low	Global	Confirmed

Description

The current comments provide a general overview of the function's purpose, but they lack depth in explaining the logic and implementation details. Enhancing the comments to clarify key steps, assumptions, and edge cases would improve readability and maintainability.

Recommendation

Add more descriptive code comments

Resolution

Acknowledged

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.