Three Sigma

Code Audit

Mitosis

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Disclaimer

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Table of Contents

Disclaimer 3
Summary 7
Scope 9
Methodology 11
Project Dashboard 13
Code Maturity Evaluation 16
Findings 19
- 3S-Mitosis-C01 19
- 3S-Mitosis-M01 20
- 3S-Mitosis-M02 21
- 3S-Mitosis-M03 22
- 3S-Mitosis-M04 23
- 3S-Mitosis-M05 24
- 3S-Mitosis-M06 25
- 3S-Mitosis-L01 26
- 3S-Mitosis-L02 27
- 3S-Mitosis-L03 28
- 3S-Mitosis-L04 29
- 3S-Mitosis-L05 30
- 3S-Mitosis-L06 31
- 3S-Mitosis-N01 32
- 3S-Mitosis-N02 33
- 3S-Mitosis-N03 34
- 3S-Mitosis-N04 35
- 3S-Mitosis-N05 36
- 3S-Mitosis-N06 37
- 3S-Mitosis-N07 38
- 3S-Mitosis-N08 39
- 3S-Mitosis-N09 40
- 3S-Mitosis-N10 41
- 3S-Mitosis-N11 42
- 3S-Mitosis-N12 43
- 3S-Mitosis-N13 44
- 3S-Mitosis-N14 45
- 3S-Mitosis-N15 46
- 3S-Mitosis-N16 47
- 3S-Mitosis-N17 48
- 3S-Mitosis-N18 49
- 3S-Mitosis-N19 50

Summary

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Summary

Three Sigma audited Mitosis in a 1.5 weeks engagement. The audit was conducted from 25-06-2024 to 05-07-2024.

Protocol Description

Mitosis is an Ecosystem-Owned Liquidity layer 1 blockchain that optimizes multi-chain liquidity provision by allowing LPs to deposit assets for miAssets, participate in governance to allocate liquidity across networks and protocols, and aims to empower retail LPs with institutional-grade opportunities while helping protocols bootstrap liquidity in a decentralized manner.

Scope

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Scope

Filepath	nSLOC
hooks/AggregateHook.sol	115
hooks/Cap.sol	242
lib/RedeemQueue.sol	270
strategy/storage/StrategyExecutorStorageV1.sol	24
strategy/StrategyExecutor.sol	161
vault/BasicVault.sol	259
vault/storage/BasicVaultStorageV2.sol	41
Total	**1112**

Assumptions

External libraries are considered safe.

Methodology

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Project Dashboard

Application Summary

Name	Mitosis
Commit	b545f6724c3f0b5eebea72c652d7f9728f1627f0
Language	Solidity
Platform	Ethereum, Arbitrum, Optimism, Manta, Mode, Blast, Linea

Engagement Summary

Timeline	25-06-2024 to 05-07-2024
№ of Auditors	2
Review Time	1.5 weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	1	1	0
High	0	0	0
Medium	6	5	1
Low	6	6	0

None	19	18	1

Category Breakdown

Suggestion	4
Documentation	0
Bug	17
Optimization	4
Good Code Practices	7

Code Maturity Evaluation

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Weak. The owner can change signers at any time.
Code Stability	Satisfactory. No new functionality was being added throughout the audit.
Upgradeability	Satisfactory. The protocol is upgradeable.
Function Composition	Satisfactory. Functionality was well split into functions.
Front-Running	Satisfactory. No significant frontrunning opportunities were found.
Monitoring	Satisfactory. Events are correctly emitted for the most significant state changes.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Satisfactory. Unit and fuzz tests were present for most functionality.

Findings

Code Audit

Mitosis Ecosystem-Owned Liquidity Layer 1 Blockchain

Findings

3S-Mitosis-C01

`BasicVault::_redeem()` burns **`newAmount`** but redeems **`amount`**, allowing attackers to drain the vault

Id	3S-Mitosis-C01
Classification	Critical
Severity	Critical
Likelihood	Medium
Category	Bug
Status	Addressed in #69d1369.

Description

`BasicVault::_redeem()` is inconsistent as it burns **`newAmount`**, the value returned from the hook, but pushes a request to the redeem queue with **`amount`**. As **`hook::reportRedeem()`** will cap the amount to redeem to the current **`load`**, the last user to withdraw may specify a huge amount to redeem, which will be capped to the remaining load, burning the capped **`newAmount`** shares, but pushing a request with the huge **`amount`**.

Recommendation

```
uint256 requestId = $v2.redeemQueue.push(_msgSender(), newAmount);
```

3S-Mitosis-M01

`BasicVault` is incompatible with fee-on-transfer tokens

Id	3S-Mitosis-M01
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Acknowledged

Description

The deposit and redeem functions of the vault take an **`amount`** parameter and mint/burn the corresponding amounts assuming a 1:1 ratio. However, fee-on-transfer tokens charge a small fee on every transfer, disrupting the 1:1 ratio and causing issues with internal accounting.

Recommendation

Compute the balance before and after the transfer, then subtract them to get the actual amount transferred. Additionally, use the nonReentrant modifier to prevent reentrancy in ERC777 tokens.

3S-Mitosis-M02

Metadata is not set in **`Cap`**, which is required by the inherited **`Router`** to specify gas limit to the mailbox

Id	3S-Mitosis-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #669b422.

Description

The gas limit of messages must be set in the metadata function, as indicated by the docs. When it is not set, the gas limit defaults to **`50_000`**, which may not be enough to handle the epoch increase on the destination.

Recommendation

Overwrite the function with the correct metadata. Additionally, the Hyperlane version used is old, consider using a more recent one. In the newer version, metadata is sent as an argument, which makes it more obvious that it must be set.

3S-Mitosis-M03

`StrategyExecutor::disableStrategy()` disables the wrong strategy

Id	3S-Mitosis-M03
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #4f60215.

Description

`StrategyExecutor::disableStrategy()` removes a strategy from the **`enabled`** array by moving the last element to the to be deleted element and popping. However, the check to move the element is incorrect, **`if (enabled.length > 1) enabled[i] = enabled[enabled.length - 1];`**. This will pop the last strategy incorrectly.

Additionally, the id to disable could be sent as an argument to save gas on the lookup and the loop can break after finding the index.

Recommendation

The element must be copied when it is not the last one (if it is the last one, it should just be popped). Thus, the correct check is

`if (i != enabled.length - 1) enabled[i] = enabled[enabled.length - 1];`.

3S-Mitosis-M04

`RedeemQueue::getAvailableResolveRange()` incorrectly returns false if **`from == count - 1`**

Id	3S-Mitosis-M04
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #87756b8.

Description

`RedeemQueue::getAvailableResolveRange()` returns the indices to resolve a requestor, but returns nothing if **`from == count - 1`**. For example, this would return when **`from == 0`** and **`count == 1`** (there's 1 element), when it should not. This could be problematic if a user only has 1 request and the redeem queue is disabled, it would never be able to claim until the redeem queue is enabled again.

Recommendation

Return if **`from == count`**. The check **`if (count == 0) return (0, 0, false);`** is also not required, as it is covered by the other check.

3S-Mitosis-M05

`BasicVault::_redeem()` does not correctly deal with a disabled redeem queue after it was enabled

Id	3S-Mitosis-M05
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #cfd54e5, #f379467.

Description

`BasicVault::_redeem()` only calls **`BasicVault::_resolveWithIdleBalance()`** when the redeem queue is enabled, but it should also do it when it is disabled as not enough assets may have been reserved. Additionally, when the redeem queue is disabled, it only checks the balance of the contract, not the **`_idleBalance()`**, as amounts may have been reserved to fulfill requests.

Recommendation

```
...
_resolveWithIdleBalance($v2, asset_);
if ($v2.redeemQueueEnabled) {
  uint256 requestId = $v2.redeemQueue.push(_msgSender(), amount);
  emit RedeemQueued(receiver, address(asset_), requestId);
} else {
  if (_idleBalance($v2, asset_) < newAmount)
    revert("You're unable to redeem your assets now. Please try again
later.");
  asset_.safeTransfer(receiver, newAmount);
}
...
```

3S-Mitosis-M06

`BasicVault::_deposit()` should always resolve with idle balance as the redeem queue may be disabled with requests pending

Id	3S-Mitosis-M06
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #edf8184.

Description

`BasicVault::_deposit()` only calls **`BasicVault::_resolveWithIdleBalance()`** if the redeem queue is enabled; however, requests must be claimed if they were created before the redeem queue was disabled, so the idle balance must be reserved even when the redeem queue is disabled.

Recommendation

Always resolve with idle balance in the **`BasicVault::_deposit()`** function.

3S-Mitosis-L01

Protocol should disable renouncing ownership if it is never intended

Id	3S-Mitosis-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #ccd98e9.

Description

Contracts inheriting **`Ownable2StepUpgradeable`** or **`Ownable`** allow for renouncing ownership, which could be catastrophic if triggered by mistake.

Recommendation

Consider overriding **`renounceOwnership()`** to revert if called, disabling this functionality.

3S-Mitosis-L02

The **`StrategyExecutor`** can remain in a paused state if the owner renounces control while it's paused

Id	3S-Mitosis-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #ccd98e9.

Description

The **`StrategyExecutor`** contract inherits **`Ownable2StepUpgradeable`** and **`PausableUpgradeable`**, allowing the owner to pause the contract. The issue arises if the contract is paused and the owner renounces ownership, leaving the contract permanently paused.

Recommendation

Consider removing the option for renouncing ownership or reimplementing the logic to ensure the contract unpauses before ownership is renounced.

3S-Mitosis-L03

`BasicVault::manualRedeem()` and `BasicVault::manualDeposit()` are inconsistent

Id	3S-Mitosis-L03
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #07b77cf.

Description

`BasicVault::manualDeposit()` does not pull assets from the `msg.sender`, but `BasicVault::manualRedeem()` burns shares from `msg.sender`, which is inconsistent. In fact, `BasicVault::manualRedeem()` is exactly the same as `BasicVault::redeem()`, the only difference being that the former is permissioned.

Recommendation

The purpose of `BasicVault::manualRedeem()` is unclear.

3S-Mitosis-L04

`BasicVault::getRedeemRequestOf()` will revert as the redeem requests are filled with the wrong indexes

Id	3S-Mitosis-L04
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #99fee9a.

Description

In **`BasicVault::getRedeemRequestOf()`**, **`redeemRequests`** is filled with the requests from the queue from the **`IndexByRequestor`** struct **`offset`**, which stores the last non resolved request. In the loop where **`redeemRequests`** is assigned, it does **`redeemRequests[i] = ...`**, but **`i`** starts at **`from`**, which may not be 0 and revert in this case.

Recommendation

Replace the code to **`redeemRequests[i - from] = ...`**.

3S-Mitosis-L05

`Cap::receive()` does not place restrictions in the **`sender`**, which may lead to donations

Id	3S-Mitosis-L05
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #b9ed378.

Description

`Cap::receive()` has no restrictions in place of the **`msg.sender`**, which means anyone can send funds to **`Cap`**, without any effect.

Recommendation

Delete the function or specify who may call it.

3S-Mitosis-L06

`_msgSender()` is mixed with `msg.sender`

Id	3S-Mitosis-L06
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #445f850.

Description

The code is using OpenZeppelin's Context contract which is intended to allow meta-transactions. It works by using a call to _msgSender() instead of querying msg.sender directly, because the method allows those special transactions. The problem is that the bridge adapters use msg.sender directly instead of _msgSender(). While this doesn't significantly impact the functions called from CCDMHost, as meta-transactions aren't needed there, it causes issues for users who call the adapters directly.

Recommendation

Change the code to use _msgSender() instead of msg.sender.

3S-Mitosis-N01

Adopt named mappings for clarity

Id	3S-Mitosis-N01
Classification	None
Category	Good Code Practices
Status	Addressed in #dfc8c2c.

Description

To enhance code clarity, consider employing named mappings.

3S-Mitosis-N02

`Linea` does not support `PUSH0`

Id	3S-Mitosis-N02
Classification	None
Category	Suggestion
Status	Addressed in #3b09b5e.

Description

`Linea` does not support `PUSH0`, so it may be necessary to compile the contracts with an old em version (such as Paris, which is the one currently used).

Recommendation

Keep in mind that the contracts will not work on Linea as is if the evm version picked is `shanghai` or more recent.

3S-Mitosis-N03

`BasicVaultFactory::createVault()` inconsistent already existing vault check

Id	3S-Mitosis-N03
Classification	None
Category	Bug
Status	Addressed in #7c8ca62.

Description

`BasicVaultFactory::createVault()` reverts with a vault already created error whenever **`$.vaultIds[vault] != 0 && $.vaults[$.vaultIds[vault]] != vault`**. This is problematic because the vault with **`id == 0`** will not revert due to this error (but still reverts later as the creation will fail) and **`$.vaults[$.vaultIds[vault]]`** is always **`vault`**, as it is impossible to have mismatching vaults and vault ids.

Recommendation

Vault ids could start at **`1`** instead so index 0 is never confused with not having been created. This way it is enough checking the vault to id mapping.

3S-Mitosis-N04

`Initializable` constracts should call **`_disableInitializers()`** in the constructor instead of using the **`initializer`** modifier

Id	3S-Mitosis-N04
Classification	None
Category	Good Code Practices
Status	Addressed in #70683f0.

Description

When locking the implementation contract, the usual behaviour is calling **`_disableInitializers()`**.

Recommendation

For better readability, consider calling **`_disableInitializers()`** instead of placing the **`initializer`** modifier in the constructor.

3S-Mitosis-N05

`pragma abicoder v2;` is turned on by default after a certain solidity version

Id	3S-Mitosis-N05
Classification	None
Category	Good Code Practices
Status	Addressed in #8b05c08.

Description

`pragma abicoder v2;` is not required anymore as it is used by default.

Recommendation

Remove **`pragma abicoder v2;`**.

3S-Mitosis-N06

`RedeemQueue::isResolved()` `resolvedCount` always returns `requestIds.length`

Id	3S-Mitosis-N06
Classification	None
Category	Bug
Status	Addressed in #9ec2322.

Description

`resolvedCount` in `RedeemQueue::isResolved()` always increments in the loop, regardless of it being `resolved_` or not.

The function is not exposed so it has no impact but could be a future issue.

Recommendation

Only increment the count when it is resolved.

3S-Mitosis-N07

`RedeemQueue::findOffsetIndex()` does not check the last index

Id	3S-Mitosis-N07
Classification	None
Category	Bug
Status	Addressed in #bed727e.

Description

`RedeemQueue::findOffsetIndex()` performs binary search but does not check the last index when doing so, as it assigns `uint256 r = len_ - 1;`. For example, consider `l == 0, len == 3`, so `r == 2` and there are requests `0`, `1` and `2`.

First iteration, `m == 0 + (2 - 0) / 2 == 1`.

`isReserved()` returns true

`l == m + 1 == 1 + 1 == 2` and it leaves the while loop as `l == r == 2`.

Thus, the last index is not checked (`m == 2`).

Recommendation

`uint256 r = len_;` would correctly check index `2` and assign `l` to 3 in this case.

Additionally, consider exposing this function in the `BasicVault` as it currently is not.

3S-Mitosis-N08

`Structs` in **`RedeemQueue`** are out of order and should all be placed at the top

Id	3S-Mitosis-N08
Classification	None
Category	Good Code Practices
Status	New Issue

Description

The style guide can be found here, and structs should not be placed in between code.

Recommendation

Follow the style guide for better readability.

3S-Mitosis-N09

`RedeemQueue` could be optimized

Id	3S-Mitosis-N09
Classification	None
Category	Optimization
Status	Addressed in #bed727e.

Description

The current **`RedeemQueue`** implementation goes through requests and marks them as resolved one by one. This is expensive and a much better solution is keeping track of the last index that was resolved, and if the request index is smaller than the last resolved, it means it was resolved (indices are sequential). Here is the function that settles redeem requests. It gets the correct index to redeem until to by perform binary search, but can also pass it as argument (so the search is performed off chain).

It's also not required to store **`request.amount`** as it can be calculated by fetching the last request and subtracting from the current cumulative amount.

Recommendation

It would be much cheaper to redeem batches at once, so a similar method to Lido's could be implemented.

3S-Mitosis-N10

`RedeemQueue::get()` reverts due to underflow when it should revert and throw the correct error

Id	3S-Mitosis-N10
Classification	None
Category	Good Code Practices
Status	Addressed in #17568a7.

Description

`RedeemQueue::get()` reverts due to underflow if **`index_.count == 0`** instead of the error **`IndexOutOfRange`**.

Recommendation

```
IndexOutOfRange(index._offset, index._count, idx);
```

3S-Mitosis-N11

Solidity types **`uint256`** are never negative

Id	3S-Mitosis-N11
Classification	None
Category	Good Code Practices
Status	Addressed in #b13e220.

Description

`BasicVault::_deposit()` checks if **`amount`** is negative, which is unnecessary.

Recommendation

Replace **`<=`** by **`==`**.

3S-Mitosis-N12

`BasicVault::_resolveWithIdleBalance()` can return before calculating `_idleBalance()` to save gas

Id	3S-Mitosis-N12
Classification	None
Category	Optimization
Status	Addressed in #a607d6c.

Description

`BasicVault::_resolveWithIdleBalance()` returns when there is enough reserved amount to fulfill all requests. In this case, there is no need to fetch the `_idleBalance()`, should the order should be inverted to save gas.

Recommendation

```
function _resolveWithIdleBalance(StorageV2 storage $v2, IERC20 asset_)
internal returns (uint256 resolved) {
  uint256 remaining = $v2.redeemQueue.remaining();
  if (remaining == 0) return 0;
  uint256 idleBalance_ = _idleBalance($v2, asset_);
  resolved = remaining > idleBalance_ ? idleBalance_ : remaining;
  $v2.redeemQueue.reserve(resolved);
  return resolved;
}
```

3S-Mitosis-N13

Storage variables can be cached to save gas

Id	3S-Mitosis-N13
Classification	None
Category	Optimization
Status	Addressed in #43e8fea.

Description

Storage variables should be cached to avoid reading from storage more than once and incurring extra gas costs.

Recommendation

In **`BasicVault::enableEOL()`**, **`$.asset.forceApprove(address($v2.eol), type(uint256).max);`** should use **`eol`** instead.

In **`RedeemQueue::push()`**, **`queue._count++;`** could be **`queue._count = requestId + 1`**.

In **`RedeemQueue::resolve()`**, **`update()`** writes the whole request to storage, when only the resolved at field is modified.

In **`StrategyExecutor::disableStrategy`**, **`enabled.length`** should be cached.

3S-Mitosis-N14

`decimals()` is not part of the **`ERC20`** standard and not all tokens may implement it as expected, which may cause **`initialize()`** to fail

Id	3S-Mitosis-N14
Classification	None
Category	Suggestion
Status	Acknowledged

Description

`BasicVault` is initialized by calling **`asset::decimals()`**, but as it is not part of the standard, it may not be implement the function in the expected way or at all.

Recommendation

Nothing may be required if the tokens to be integrated all implement the decimals function as is. Alternatively, decimals could be passed as argument.

3S-Mitosis-N15

`Cap::_checkRemoteStateAndAdvance()` may be optimized by returning early as soon as a different epoch is found in a remote domain

Id	3S-Mitosis-N15
Classification	None
Category	Optimization
Status	Addressed in #40ef30e.

Description

`Cap::_checkRemoteStateAndAdvance()` advances the next epoch if all domains are equal to the minimum domain found. Thus, if any of the domains is not equal to the minimum, it will return anyway.

Recommendation

Given that it returns if any of epochs of the domains is different, there is no need to search for the minimum. It's enough picking the domain of index 0 and returning if any of the other domains is not the same. It later only advances the epoch if the epoch of the domains is bigger than the local, so if they all match the current local domain, it will not advance anyway.

3S-Mitosis-N16

The domain with 0 index in **`Cap::_checkRemoteStateAndAdvance()`** is not checked for equality

Id	3S-Mitosis-N16
Classification	None
Category	Bug
Status	Addressed in #40ef30e.

Description

`Cap::_checkRemoteStateAndAdvance()` returns if not all domains have the same epoch, but the domain with index 0 is skipped in the loop. This is never a problem because the only way for it not to return is if the domains are something such as **`2, 1, 1`**. However, as it only advances the epoch if the minimum epoch is bigger than the local domain's epoch, as the current is 1 and the minimum is 1, it will do nothing.

Recommendation

Consider starting the loop in index 0.

3S-Mitosis-N17

The current epoch will never advance if there are no other domains

Id	3S-Mitosis-N17
Classification	None
Category	Suggestion
Status	Addressed in #67aa296.

Description

In **`Cap::_processDone()`**, it returns if no domains are set, **`if (domains.length == 0) return;`**. This means it will be impossible to advance epoch if there are no domains set, which may be fine as the cap can be increased either way.

Recommendation

Consider adding a comment with the intended behaviour or modify it if required.

3S-Mitosis-N18

`Cap::setEpochCap()` may add a stale `cap`

Id	3S-Mitosis-N18
Classification	None
Category	Bug
Status	Addressed in #ef1aa9a.

Description

`Cap::setEpochCap()` reverts when `nextCap < $.cap[i]`, allowing the next cap to be equal to the current cap, which does not seem intended according to the error reason 'cap should be greater than previous cap'.

Recommendation

Replace the inequality check with less or equal to.

3S-Mitosis-N19

In **`AggregateHook`**, several **`if (newAmount == 0)`** checks are ambiguous

Id	3S-Mitosis-N19
Classification	None
Category	Good Code Practices
Status	Addressed in #b9b9ef0.

Description

In **`AggregateHook::reportRedeem()`**, **`AggregateHook::previewReportClaim()`** and **`AggregateHook::reportClaim()`**, it sets **`newAmount`** to **`prevAmount`** if **`newAmount`** is null, which is always, as it is initialized by default to null.

Recommendation

Modify the code to make the intent more clear.