Audit Report

Lido Finance – stLuna

v1.2

November 23, 2021

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by Lido Finance to perform a security audit of stLuna

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/lidofinance/anchor-bAsset-contracts/commit/559d18739cfede1655ded1d4d5741464b0fdd47b

Commit hash: `e124bf8e3d8220e08f2fade196cb5b22e699bb61`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Lido Finance is seeking to augment the original Terra bAsset design by allowing stakers to reinvest staking rewards into a staked version of bLuna called stLuna, as well as decouple validation to an approach that rewards validators for efficiency while attempting to maintain an equal distribution of stake across validators.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Slashed validators may cause unbonding to fail	Critical	Resolved
2	Missing tax deduction leads to users losing their bond	Critical	Acknowledged
3	Lack of applying bLuna recovery fee to the exchange rate during stLuna bLuna conversion will lose user tokens	Critical	Resolved
4	Condition on bLuna/stLuna undelegation amount can lead to an error that blocks any undelegation	Critical	Resolved
5	Undelegation waitlist is unbounded which may result in users being unable to withdraw unbonded tokens	Major	Acknowledged
6	Rewards are distributed proportional to stLuna/bLuna tokens, not proportional to bonded Luna tokens	Major	Resolved
7	Rewards distribution fails if no tokens are bonded for bLuna	Major	Resolved
8	Conversion between bLuna and stLuna or vice versa likely applies incorrect peg recovery fee	Major	Resolved
9	Updates to the hub's bLuna and stLuna contract addresses may lead to incorrect undelegation amounts	Major	Resolved
10	Manually unpausing the hub contract after migration might lead to inconsistent state	Minor	Resolved
11	Usage of raw queries could break functionality in the future	Minor	Resolved
12	Calculation of required peg fee relies on threshold being less than or equal to one, but that validation is missing	Minor	Resolved
13	Changes to stLuna/bLuna reward denom could cause balances to become inaccessible	Minor	Resolved
14	Lack of validation of peg recovery fee param on	Minor	Resolved

	hub contract may lead to errors during bonding, unbonding or token conversion		
15	Hub contract's state query returns outdated exchange rates	Minor	Resolved
16	Hub's unbonding period parameter will lead to users receiving less than bonded tokens if set incorrectly	Minor	Acknowledged
17	Ability to pause hub contract increases risks associated with compromised owner key	Informational	Acknowledged
18	Users are subject to slashing between unbonding and undelegation batch execution, which is currently not documented	Informational	Acknowledged
19	Canonical address transformations are inefficient	Informational	Acknowledged
20	Duplicate token supply queries during the hub's bond message are inefficient	Informational	Acknowledged
21	Duplicate exchange rate queries in rewards dispatcher are inefficient	Informational	Resolved
22	Token contract initialization structs contain unused fields which is inefficient	Informational	Resolved
23	Storing of total delegated amount in validator registry is inefficient	Informational	Resolved
24	stLuna token contract could be replaced by CW20 base contract	Informational	Acknowledged
25	Last index modification state and reward denom param of hub contract are unused	Informational	Acknowledged
26	Hub contract state in stLuna token contract is unused	Informational	Resolved
27	Remove validator implementation is inefficient	Informational	Resolved
28	Using nested loops to find validator delegations is inefficient	Informational	Resolved
29	Hub contract loads state multiple times in several message handlers which is inefficient	Informational	Resolved
30	Unused function argument	Informational	Resolved
31	LastBatch type in hub contract is unused	Informational	Resolved

32	Inefficient reward calculation	**Informational**	**Resolved**
33	Inefficient storage iteration during hub’s withdraw unbonded function	**Informational**	**Resolved**
34	Ignoring negative case of signed integer subtraction is bad practice	**Informational**	**Resolved**

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium-High**	-
Code readability and clarity	**Medium**	-
Level of Documentation	**Medium-High**	-
Test Coverage	**Medium-High**	-

Detailed Findings

1. Slashed validators may cause unbonding to fail

Severity: Critical

The subtraction in the `calculate_undelegations` function in `contracts/anchor_basset_validators_registry/src/common.rs:70` may underflow and hence cause a panic if a validator has fewer `total_delegated` tokens than the calculated `coins_per_validator` plus the potential extra coin. That could happen for example if a validator got slashed. Such a panic would prevent users from unbonding.

Recommendation

We recommend changing the `calculate_undelegations` function to not just reduce validators' delegations, but rather even them out. Alternatively, a checked subtraction with the handling of the underflow by unbonding from another validator could be implemented.

Status: Resolved

2. Missing tax deduction leads to users losing their bond

Severity: Critical

During the hub's `WithdrawUnbonded` message handler, tax is not deducted in `contracts/anchor_basset_hub/src/unbond.rs:155-157`. That implies that the hub's balance of `coin_denom` will be used to pay taxes. The last user trying to withdraw their tokens will not be able to withdraw them as there will not be enough tokens left in the contract.

Recommendation

We recommend deducting the tax from the sent amount in `contracts/anchor_basset_hub/src/unbond.rs:157` to maintain the integrity of funds in the contract.

Status: Acknowledged

Lido Finance acknowledges this issue, but states that there is no intention to instantiate the hub contract with any denom other than LUNA. Since LUNA is not subject to taxation, this issue is not applicable for the intended use case.

3. Lack of applying bLuna recovery fee to the exchange rate during stLuna bLuna conversion will lose user tokens

Severity: Critical

The `convert_stluna_bluna` function of the hub contract triggers an update of the bLuna exchange rate without considering the fee in `contracts/anchor_basset_hub/src/convert.rs:61`: The exchange rate is updated by increasing the bluna supply by `bluna_to_mint` while the actually minted amount is `bluna_mint_amount_with_fee`, which is reduced by the peg recovery fee. Whenever a recovery fee is applied, `bluna_mint_amount_with_fee` will be smaller than `bluna_to_mint`, resulting in an exchange rate that is smaller than it should be. That will result in every token holder losing tokens. Also, the `bluna_amount` attr in line `92` is incorrectly using `bluna_to_mint`.

Recommendation

We recommend using `bluna_mint_amount_with_fee` in `contracts/anchor_basset_hub/src/convert.rs:61` and in line `92`.

Status: Resolved

4. Condition on bLuna/stLuna undelegation amount can lead to error that blocks any undelegation

Severity: Critical

The `process_undelegations` function currently returns an error when the `bluna_undelegation_amount` or the `stluna_undelegation_amount` equals one in `contracts/anchor_basset_hub/src/unbond.rs:473`. That error will abort the `execute_unbond` or `execute_unbond_stluna` functions when the `epoch_period` has been passed in such a state.

An attacker could simply wait for an empty undelegation batch for either bLuna or stLuna tokens, and then unbond one unit of the respective token.

If the contract ends in such a state, the only way to recover would be to drastically change the exchange rate such that the `bluna_undelegation_amount` or the `stluna_undelegation_amount` equals zero or two, which is very costly.

Recommendation

We recommend removing the condition in `contracts/anchor_basset_hub/src/unbond.rs:470-471` or, alternatively, returning `Ok` instead of an `Err` in line `473`.

Status: Resolved

5. Undelegation waitlist is unbounded which may result in users being unable to withdraw unbonded tokens

Severity: Major

In the `get_unbond_batches` function in `contracts/anchor_basset_hub/src/state.rs:97`, an unbounded iteration is performed over all unbond batches for a user that tries to withdraw their tokens using the hub's `WithdrawUnbonded` message. In the unlikely event that a user has a very high amount of batches, the iteration may take more gas than a single message is allowed to consume. That would effectively block a user from retrieving their tokens.

Despite a potential loss of access to tokens, we classify this issue as major, since the likelihood of a user having many stored unbond batches is very low. There may be scenarios with high unbond batches per user though, for example, if another contract builds on top of stLuna, triggering many small bonds, and then a market crash triggering many unbonds.

Recommendation

We recommend adding an optional limit parameter to the `WithdrawUnbonded` message to allow users to withdraw their funds independent of the number of unbond batches. Likewise, we recommend adding an optional limit parameter to the `WithdrawableUnbonded` query. That limit parameter should be applied to the unbounded queries in `contracts/anchor_basset_hub/src/state.rs:102`, `124` and `148`. Alternatively, a limit could be put on the number of unbond batches a user can store.

Status: Acknowledged

Lido Finance acknowledges this issue, but states that it is unlikely to cause problems before the year 2025. The team has added the issue to their backlog and plans to upgrade the contracts before it becomes problematic.

6. Rewards are distributed proportional to stLuna/bLuna tokens, not proportional to bonded Luna tokens

Severity: Major

The logic in `contracts/anchor_basset_rewards_dispatcher/src/contract.rs:291` implies a reward distribution proportional to minted stLuna and bLuna. Since bLuna is pegged and should recover its exchange rate over time, but stLuna is not, this will imply a reward distribution that fluctuates with the exchange rate between stLuna and bLuna over time.

For example, imagine there are 1000 tokens bonded for stLuna, and 1000 tokens bonded for bLuna, but assume that due to slashes, the stLuna/Luna exchange rate is broken such that 1 stLuna is only worth 0.5 Luna tokens, while the bLuna exchange rate got recovered to 1 Luna token per bLuna. The current logic assigns from 300 Luna staking rewards 200 to bLuna, and only 100 to stLuna, even though the staked amounts are the same.

Recommendation

We recommend changing the logic to use the bonded tokens for proportional distribution, rather than the issued bLuna/stLuna tokens.

Status: Resolved

7. Rewards distribution fails if no tokens are bonded for bLuna

Severity: Major

The hub's `UpdateGlobalIndex` message sends the `UpdateGlobalIndex` message to the reward contract, which will return an error if the amount of bonded Luna tokens towards bLuna is zero in `contracts/anchor_basset_reward/src/global.rs:84`. That error will prevent any reward distribution.

Recommendation

We recommend returning `Ok` instead of an error in `contracts/anchor_basset_reward/src/global.rs:84`.

Status: Resolved

8. Conversion between bLuna and stLuna or vice versa likely applies incorrect peg recovery fee

Severity: Major

When converting between bLuna and stLuna or vice versa, the peg recovery fee is applied in `contracts/anchor_basset_hub/src/convert.rs:44` or `131`. There is no update of the exchange rate though to account for potential slashes before that fee application, which implies that the peg recovery fee is likely incorrect.

Recommendation

We recommend calling the `slashing` function to update the exchange rates before applying the peg recovery fee.

Status: Resolved

9. Updates to the hub's bLuna and stLuna contract addresses may lead to incorrect undelegation amounts

Severity: Major

The hub contract currently allows config updates to the `bluna_token_contract` and `stluna_token_contract` config values in `contracts/anchor_basset_hub/src/config.rs:93` and `102`. Such updates can cause wrong amounts of tokens to be calculated during processing of undelegations, since the amount of unbonded bLuna/stLuna tokens is stored for batched unbonding as `requested_bluna_with_fee` and `requested_stluna`.

Recommendation

We recommend removing the ability to update `bluna_token_contract` and `stluna_token_contract` config values, or asserting that `requested_bluna_with_fee` and/or `requested_stluna` is zero before allowing an update of the contracts' addresses

Status: Resolved

10. Manually unpausing the hub contract after migration might lead to inconsistent state

Severity: Minor

The migration process described below has been implemented after the frozen commit for this audit. A change to the migration logic of the hub contract was necessary since a migration within one transaction ran out of gas. An audit of the updated migration logic has been performed on commit `e04eb1313c481bdeae084a1dab064afdab5ddbae`.

During migration of the hub contract, a `paused` param is set to true in `contracts/anchor_basset_hub/src/contract.rs:755`, which leads to a rejection of all messages except updates of params and the migration of old unbond waitlist entries in line `137`.

In the current implementation, the contract must be explicitly unpaused by the owner when the migration is done. There is no validation that the migration has actually been fully performed though. If unpaused before the migration has been finished, partially migrated unbonding waitlist entries might be processed, which could cause missed waitlist entries without a way to recover.

Additionally, the paused flag can be used at any time by the owner to pause/unpause the hub contract. If the owner key is ever compromised, ownership could be transferred and the contract paused, leaving any funds inaccessible.

We classify this issue as minor since it can only be caused by the owner.

Recommendation

We recommend changing the migration logic into a state machine to guarantee consistency of migrations. This could be achieved by not just automatically setting a `migration_pending` boolean when the migration process starts, but by also automatically removing the `migration_pending` boolean once the old unbond wait list has no more entries and the migration is done. This change also removes the risk of a compromised owner key locking the contracts.

Status: Resolved

11. Usage of raw queries could break functionality in the future

Severity: Minor

In several places in the code base, raw queries are used. Raw queries tie the querier to the queried contract's storage layout. This can be problematic because changes to the storage layout are not normally considered breaking changes and are not well documented.
Raw queries are currently used in: `contracts/anchor_basset_token/src/querier.rs:11`, `23`, `contracts/anchor_basset_reward/src/querier.rs:8` and line `21`.

Recommendation

We recommend using smart queries instead of raw queries.

Status: Resolved

12. Calculation of required peg fee relies on threshold being less than or equal to one, but that validation is missing

Severity: Minor

The `required_peg_fee` calculation in `contracts/anchor_basset_hub/src/bond.rs:79` requires the `er_threshold` param to be less than or equal to one. If that does not hold, the calculation in line `81` may underflow and cause a panic. There is no validation of that condition in the contract's `instantiate` and `execute_update_params` functions.

Recommendation

We recommend adding validation to the contract's `instantiate` and `execute_update_params` functions to ensure that `er_threshold` is less than or equal to one.

Status: Resolved

13. Changes to stLuna/bLuna reward denom of the rewards dispatcher could cause balances to become inaccessible

Severity: Minor

If the `stluna_reward_denom` or `bluna_reward_denom` is changed through `contracts/anchor_basset_rewards_dispatcher/src/contract.rs:126` or `133` between a swap and a dispatch, i. e. in the period where the contract holds a balance in the previous denom, then that balance becomes inaccessible.

We only classify this issue as minor since the funds can be retrieved by changing the denom back.

Recommendation

We recommend not to allow updates of `stluna_reward_denom` or `bluna_reward_denom`.

Status: Resolved

14. Lack of validation of peg recovery fee param on hub contract may lead to errors during bonding, unbonding or token conversion

Severity: Minor

The `peg_recovery_fee` param of the hub contract can take any `Decimal` value. If it is greater than one, bonding may return an error in `contracts/anchor_basset_hub/src/bond.rs:83`, unbonding may return an error in `contracts/anchor_basset_hub/src/unbond.rs:49` and converting bLuna to stLuna and vice versa may return an error in `contracts/anchor_basset_hub/src/convert.rs:48` or `135`.

Recommendation

We recommend adding validation to the contract's `instantiate` and `execute_update_params` functions to ensure that `peg_recovery_fee` is less than or equal to one.

Status: Resolved

15. Hub contract's state query returns outdated exchange rates

Severity: Minor

The exchange rates returned by the hub contract's state query in `contracts/anchor_basset_hub/src/contract.rs:576` and `577` are not re-fetched, so they could miss potential slashing for direct token burns from users. That might lead users to base decisions on outdated data.

Recommendation

We recommend always fetching the current exchange rates and removing the `bluna_exchange_rate` and `stluna_exchange_rate` entries from storage. Alternatively, we recommend calling the `slashing` function to update the exchange rates before returning the state.

Status: Resolved

16. Hub's unbonding period parameter will lead to users receiving less than bonded tokens if set incorrectly

Severity: Minor

The hub's config contains a parameter `unbonding_period` in `packages/basset/src/hub.rs:185`. If that value is set to less than the underlying blockchain's actual unbonding period, users might get less or even no tokens for their bonds. That is due to the `process_withdraw_rate` function which adjusts the amount of returned tokens proportionally to the ones that have been unbonded.

Recommendation

We recommend ensuring that the unbonding period is long enough.

Status: Acknowledged

Lido Finance considers adding this check once staking parameters can be queried from Terra.

17. Ability to pause hub contract increases risks associated with compromised owner key

Severity: Informational

The logic described below has been added after the frozen commit for this audit. A change to the migration logic of the hub contract was necessary since a migration within one transaction ran out of gas. An audit of the updated migration logic has been performed on commit `e04eb1313c481bdeae084a1dab064afdab5ddbae`.

To perform storage migration after an upgrade of the hub contract, a `paused` param has been added to the hub contract. Besides its intended use case to pause the hub until the migration of unbond waitlist entries has been performed, it can also be set/unset at any time by the owner to pause/unpause the hub contract in `contracts/anchor_basset_hub/src/config.rs:59`. If the owner key is ever compromised, ownership could be transferred and the contract paused, leaving any funds inaccessible.

We classify this issue as informational since a compromised owner key has other severe implications and proper key management is an underlying assumption of the protocol in any case.

Recommendation

We recommend removing the ability to pause/unpause the contract, but rather using a state machine as described above in Manually unpausing the hub contract after migration might lead to inconsistent state.

Status: Acknowledged

18. Users are subject to slashing between unbonding and undelegation batch execution, which is currently not documented

Severity: Informational

The current implementation of Lido finance does undelegations in batches for efficiency reasons. Undelegations happen at most every `epoch_period` through the logic in `contracts/anchor_basset_hub/src/unbond.rs:74`.

That implies that users that have sent unbond messages to the hub contract will still be subject to slashing until the undelegation batch is executed. This behaviour is different from Terra's/Cosmos SDK's slashing module, which only slashes delegators that were active when the infraction happened.

This difference is currently not documented.

Recommendation

We recommend documenting that undelegated funds in Lido finance will still be subject to slashing until the undelegation is executed, which can only happen after the current epoch ends.

Status: Acknowledged

Lido Finance intends to update the documentation before the release.

19. Canonical address transformations are inefficient

Severity: Informational

While previously recommended as a best practice, usage of canonical addresses for storage is no longer encouraged. The background is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit.

Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend removing any transformation from human to canonical addresses.

Status: Acknowledged

20. Duplicate token supply queries during the hub's bond message are inefficient

Severity: Informational

During the `execute_bond` function, `query_total_bluna_issued` / `query_total_stluna_issued` is executed as part of slashing call in `contracts/anchor_basset_hub/src/contract.rs:372 / 373`, and then again in `contracts/anchor_basset_hub/src/bond.rs:66 / 68`. Running the same query multiple times leads to unnecessary gas consumption.

Recommendation

We recommend only querying the total bLuna/stLuna issued once and passing it into the slashing function.

Status: Acknowledged

21. Duplicate exchange rate queries in rewards dispatcher are inefficient

Severity: Informational

In `contracts/anchor_basset_rewards_dispatcher/src/contract.rs:265`, the exchange rate of `denom_b` is queried in `denom_a`, and then in line `269`, the exchange rate of `denom_a` is queried in `denom_b`. This is unnecessary since the query will simply return the inverse.

Recommendation

We recommend only querying one of the exchange rates, and then calculating the other by inverting the ratio.

Status: Resolved

22. Token contract initialization structs contain unused fields which is inefficient

Severity: Informational

The `TokenInitMsg` structs of the stLuna and bLuna token contracts in `contracts/anchor_basset_token_stluna/src/msg.rs:7` and `contracts/anchor_basset_token/src/msg.rs:11` contain the unused `mint` fields.

Unused fields lead to unnecessary gas consumption and may confuse users.

Recommendation

We recommend removing unused fields.

Status: Resolved

23. Storing of total delegated amount in validator registry is inefficient

Severity: Informational

The `REGISTRY` state of the validator registry contract includes a `total_delegated` field for each stored validator in `contracts/anchor_basset_validators_registry/src/registry.rs:21`. Those stored values are never used – they are always fetched freshly from the underlying blockchain. Storing values that are not used is inefficient.

Recommendation

We recommend storing the validator's address only.

Status: Resolved

24. stLuna token contract could be replaced by CW20 base contract

Severity: Informational

The stLuna token contract in `contracts/anchor_basset_token_stluna/src/contract.rs` does not add or remove any functionality over `cw20_base`. Wrapping the CW20 base contract leads to more code to maintain.

Recommendation

We recommend instantiating a CW20 base token directly to reduce the amount of code to maintain.

Status: Acknowledged

Lido Finance decided to keep the wrapped version for simpler extendibility.

25. Last index modification state and reward denom param of hub contract are unused

Severity: Informational

The hub's state contains a `last_index_modification` field that is written to in `contracts/anchor_basset_hub/src/contract.rs:307`, but never read. Likewise, the hub's `reward_denom` param is unused. Storing unused data leads to increased gas cost.

Recommendation

We recommend removing unused state and params.

Status: Acknowledged

Lido Finance decided to keep the last index modification in state for better indexing.

26. Hub contract state in stLuna token contract is unused

Severity: Informational

The stLuna token contract defines a `HUB_CONTRACT` state in `contracts/anchor_basset_token_stluna/src/state.rs:4` that is written but never read. Storing unused data leads to increased gas cost without benefits.

Recommendation

We recommend removing unused state.

Status: Resolved

27. Remove validator implementation is inefficient

Severity: Informational

Iteration over all validators happens twice in the `remove_validator` function: in `contracts/anchor_basset_validators_registry/src/contract.rs:119` and in the function `query_validators` in line `140`. It would be more efficient to do it in one call.

Likewise, querying delegations happens twice in `contracts/anchor_basset_validators_registry/src/contract.rs:135` and then again in the function `query_validators` in line `140`. As before, it would be more efficient to query delegations only once.

Recommendation

We recommend removing the iteration over validators in line `119` and assert the count below, or pass the iterator to remove an additional storage read. We also recommend querying delegations once and passing them to `query_validators`.

Status: Resolved

28. Using nested loops to find validator delegations is inefficient

Severity: Informational

Usage of a nested loop in `contracts/anchor_basset_validators_registry/src/contract.rs:227` is less efficient than using a map to access validator delegations.

Recommendation

We recommend using a map for delegation lookup. This will increase efficiency from `N*M` iterations to `N+M` iterations, where `N` is the number of validators in the registry, and `M` is the number of delegations.

Status: Resolved

29. Hub contract loads state multiple times in several message handlers which is inefficient

Severity: Informational

Within the hub contract, state is loaded multiple times in several message handlers. This is inefficient since every storage read consumes gas. Instances are:

- `contracts/anchor_basset_hub/src/bond.rs:59` and within the `slashing` function called in `61`
- `contracts/anchor_basset_hub/src/unbond.rs:35` and within the `slashing` function called in `37`

Recommendation

We recommend refactoring these instances to return an already loaded state or pass state into the `slashing` function as a mutable variable.

Status: Resolved

30. Unused function argument

Severity: Informational

The `convert_to_target_denoms` function accepts an unused `_contr_addr` argument in `contracts/anchor_basset_rewards_dispatcher/src/contract.rs:227` that leads to an unnecessary increase in the contract code size.

Recommendation

We recommend removing unused arguments.

Status: Resolved

31. LastBatch type in hub contract is unused

Severity: Informational

The hub contract defines a `LastBatch` type in `contracts/anchor_basset_hub/src/state.rs:14` that is unused and leads to an unnecessary increase in the contract code size.

Recommendation

We recommend removing unused types.

Status: Resolved

32. Inefficient reward calculation

Severity: Informational

The current reward calculation of bLuna rewards in `contracts/anchor_basset_reward/src/user.rs:42-43` uses string conversions, which is not efficient.

Recommendation

We recommend changing the current code to:

```
let rewards = all_reward_with_decimals * Uint128::new(1);
let decimals = all_reward_with_decimals - rewards;
```

Status: Resolved

33. Inefficient storage iteration during hub's withdraw unbonded function

Severity: Informational

During the `execute_withdraw_unbonded` function of the hub contract in `contracts/anchor_basset_hub/src/unbond.rs:110`, both the `get_finished_amount` and the `get_unbond_batches` functions are called. They both iterate over the same storage by `PREFIX_WAIT_MAP` in `contracts/anchor_basset_hub/src/state.rs:102` and `124`. That leads to unnecessary gas consumption.

Recommendation

We recommend only iterating over the storage once.

Status: Resolved

34. Ignoring negative case of signed integer subtraction is bad practice

Severity: Informational

The case of a negative result of the subtractions in `contracts/anchor_basset_hub/src/unbond.rs:251` and `281` is silently ignored. While the values should never be negative, it is considered best practice to panic if they are.

Recommendation

We recommend either using unsigned types or adding an assertion for a positive value.

Status: Resolved