Audit Report

Stader

v1.0

December 6, 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 Terraform Labs to perform a security audit of Stader.

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/stader-labs/stader-protocol-v0

Commit hash: `14530d71dc67c55795e9e753b80c8f50d1190847`

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

Stader provides end-to-end management of staking delegations by allowing users to deposit funds to actively managed pools of validators. Several reward strategies will be offered eventually, the audited version of the contracts involves a single strategy: auto compounding.

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	Lack of propagation of slashes to user deposits may lead to users rushing to unbond since last unbonding users may not be able to receive their stake back	Critical	Resolved
2	Validator removal happens without updating the pools contract, which leads to inconsistent state and implies that removed validators will still receive delegations	Critical	Resolved
3	Rewards are double counted, leading to slashing funds being distributed as rewards	Critical	Resolved
4	First user to deposit rewards into an empty strategy that contains remainder tokens will lose these rewards	Critical	Resolved
5	Multiple reward coins sent to SIC auto-compound and base contracts or coins sent in wrong denom will be locked inaccessibly in contract	Critical	Resolved
6	Undelegating from a strategy with not sufficient funds will lead to users losing (part of) their rewards	Critical	Resolved
7	Replacing a validator that is receiving a redelegation will lead to loss of delegation	Critical	Resolved
8	Usage of stored rather than queried delegation amounts may lead to an inconsistent state when unbonding	Major	Resolved
9	Util functions iterating over unordered hash maps may lead to Terra node consensus failures	Major	Resolved
10	Reward swap and transfer does not consider spread, which leads to distribution of slashing funds as rewards	Major	Resolved
11	Mismatch of expected and received airdrop amount can lead to lower airdrops for last users withdrawing them	Major	Resolved
12	Detached airdrop pointers update might lead to	Major	Resolved

	users receiving less airdrops than expected		
13	Discretionary reward and airdrop allocation in delegator contract might lead to incorrect accounting of rewards and airdrops and lower amount of rewards available for withdrawals	Major	Resolved
14	Rewards sent to a strategy are lost if strategy's get total tokens query returns an error	Major	Resolved
15	Compromised SCC manager keys could prevent undelegation and fetching undelegated rewards from strategies	Major	Resolved
16	Undelegation from SIC auto-compound contract may fail after slashing	Major	Resolved
17	Swapping contracts with new ones without state migrations may cause problems with funds withdrawal	Major	Resolved
18	Missing tax deduction may lead to usage of user funds, slashing compensation reserves or rewards for taxes	Minor	Resolved
19	If Terra were to change the 21 day unbonding period, Stader would run into slashing accounting errors if Stader continues using the 21 day unbonding period	Minor	Acknowledged
20	Validator stake state duplication is error-prone and inefficient	Minor	Resolved
21	Validator assignment for new deposits does not account for rewards and slashing	Minor	Resolved
22	Contract lookup by address in stader hub might lead to callers running out of gas	Minor	Resolved
23	Lack of address validation might cause errors when using invalid stored addresses	Minor	Acknowledged
24	Protocol fee values of more than 100% might cause panics	Minor	Resolved
25	Compromised manager account can take possession of user funds through airdrop claim functions which accept arbitrary messages from manager	Minor	Resolved
26	Unbounded iteration over (reward) undelegation batches could run out of gas, leaving funds inaccessible	Minor	Resolved

27	Multiple entries in pool pointers in delegator contract's reward and airdrops allocation function might lead to overwrites	Minor	Resolved
28	Transferring undelegated rewards from the SIC base does not update state	Minor	Resolved
29	The check equal coin vector util function fails to detect differences in vectors with duplicates	Minor	Acknowledged
30	Lack of validating of default user portfolio may lead to users receiving free tokens	Minor	Resolved
31	Lack of validation of default fallback strategy may lead to rewards being stuck in SCC contract	Minor	Resolved
32	Lack of check that fallback strategy is active may lead to rewards assigned to inactive fallback strategy	Minor	Resolved
33	Unbounded undelegation records may lead to loss of user funds	Minor	Resolved
34	SIC auto-compound contract does not assert that manager seed funds have been sent, could cause users getting less tokens than bonded when undelegating	Minor	Resolved
35	SIC auto-compound contract does not swap rewards before reinvesting	Minor	Acknowledged
36	Reward reinvestment in SIC auto-compound contract does not exclude jailed validators and does not even out differences in delegations	Minor	Resolved
37	Different casing of name and address of retain rewards strategy may break other protocols	Minor	Resolved
38	Stader bonds new deposits to validator with smallest stake and unbonds from validator with biggest stake, rather than evenly distributing the stake across a pool, leading to lower diversification	Informational	Acknowledged
39	Users are subject to slashing between reward unbonding and fetching of unbonded rewards	Informational	Acknowledged
40	Updating user portfolios in SCC contract does not trigger re-allocation	Informational	Acknowledged
41	Stader hub's query function is unbounded	Informational	Resolved
42	Contracts store addresses to other contracts explicitly instead of using Stader hub	Informational	Acknowledged

43	Choosing validators to undelegate from is inefficient	Informational	Resolved
44	Overflow checks not set for release profile in packages/stader-utils/Cargo.toml	Informational	Resolved
45	Validator contract's redelegate function grants call permission to pools contract, which never calls redelegate	Informational	Resolved
46	Separate exchange rate queries are inefficient	Informational	Acknowledged
47	Validator removal reply uses magic number as ID	Informational	Resolved
48	Share to token ratio uses magic number as default value	Informational	Resolved
49	Unused code impacts maintainability	Informational	Resolved
50	Duplicated code impacts maintainability	Informational	Resolved

Code Quality Criteria

Criteria	**Status**	**Comment**
Code complexity	Medium-High	Functionality is separated into different contracts, but some state is duplicated across contracts instead of fetched from a single source of truth. The functionality in the pools and validators contracts is very similar to the functionality in the sic-auto-compound contract. Furthermore, the current implementation separates the transfer of funds from updates of state into separate messages, which increases the complexity.
Code readability and clarity	Medium	-
Level of Documentation	Medium-High	-
Test Coverage	High	-

Detailed Findings

1. Lack of propagation of slashes to user deposits may lead to users rushing to unbond since last unbonding users may not be able to receive their stake back

Severity: Critical

The current architecture does not propagate slashes to user deposits, but rather relies on a slashing fund reserve to replenish slashes. That slashing fund is managed by the contract manager. There is currently no logic that reduces any user's funds to account for slashes. That makes the whole protocol inherently unstable – any slashing compensation fund will eventually run out of tokens, at which point the last users to undelegate their funds will lose those funds. Rational users will anticipate that loss, which might trigger them to rush to unbond as quickly as possible. The only way to prevent this from happening is a sufficient slashing fund reserve, but a rational manager would anticipate that rush and not be incentivized to replenish the reserves.

The current slashing compensation is implemented as follows: The validator contract's manager can add or remove slashing funds at any time through the `add_slashing_funds` and `remove_slashing_funds` functions. Those funds are tracked in the contract's state in the `slashing_funds` field. Whenever rewards are redeemed through the pools contract, the delegations of all validators are queried and a check for slashed validators is performed. If a validator got slashed, the contract uses those `slashing_funds` to delegate the slashed amount back to the validator in `contracts/validator/src/contract.rs:309`. Independent of slashing, users get their full deposited stake back on undelegation in `contracts/delegator/src/contract.rs:297`.

As described above, this process makes the Stader protocol unstable. Apart from that, there is another issue: Slashed validators are jailed on Terra, so compensating the slashed amount will not be productive.

Recommendation

We recommend changing the architecture to propagate slashes to user deposits.

Status: Resolved

Stader introduced slashing pointers in commit `ba8448c5fe2a8c75188c55c1817adff0e52411a9` to propagate slashes to users. The fix relies on regular submission of the `ReconcileFunds` message to the pools contract for proper accounting of slashes. If multiple undelegation batches are not reconciled yet, slashes will be equally distributed across them. If more than ten undelegation batches are not reconciled, slashing may roll over to users unbonding in a later batch. Stader intends to run off-chain services to submit the `ReconcileFunds` message regularly. Since that message is permissionless, any user can submit it.

2. Validator removal happens without updating the pools contract, which leads to inconsistent state and implies that removed validators will still receive delegations

Severity: Critical

Currently, the only way to remove validators is through the `remove_validator` function in the validator contract in `contracts/validator/src/contract.rs:164`. That implies that the validator storage of pools contract will not be updated: Validators are neither removed from the `validators` vector of `PoolRegistryInfo`, nor are they removed from the `VALIDATOR_REGISTRY`. Since removing a validator also leads to a re-delegation of the bonded tokens, these stake changes are not reflected in the `VALIDATOR_REGISTRY` of the pools contract.

This has several critical implications: The validator state will be inconsistent between contracts, unbonding through the pools contract will try to unbond from an already unbonded validator, and new deposits through the pools contract will be delegated to a previously removed validator through `contracts/pools/src/contract.rs:231`.

Recommendation

We recommend implementing removal of validators through the pools contract to keep the state consistent.

Status: Resolved

3. Rewards are double counted, leading to slashing funds being distributed as rewards

Severity: Critical

In the validator contract, each validator is stored in the `VALIDATOR_REGISTRY` with their current stake and their accrued rewards. That `accrued_rewards` field is incremented in different places in the contract, leading to double-counting of rewards – instances are `contracts/validator/src/contract.rs:238`, `321`, `394`, `410`, `490`, `691`, and `962`. Consequently, the total calculated in `contracts/validator/src/contract.rs:605` will be bigger than the actually collected rewards. That implies that the `swap_and_transfer` function transfers too many rewards, distributing funds as rewards that should be reserved for slashing compensation. It will also lead to the `transfer_reconciled_funds` function not returning the error in line `770` even if it is expected to.

Recommendation

We recommend changing the architecture by removing the tracking of rewards and instead simply redeeming rewards to a separate collector contract, which can then receive a message to distribute the rewards to the SCC contract. Alternatively, we recommend only adding to `accrued_rewards` the rewards that are actually redeemed in `contracts/validator/src/contract.rs:331`, which can be accomplished by querying the reward balance before and after redeeming and storing the difference.

Status: Resolved

4. First user to deposit rewards into an empty strategy that contains remainder tokens will lose these rewards

Severity: Critical

In `contracts/scc/src/helpers.rs:79`, the `get_strategy_shares_per_token_ratio` function returns the `default_s_t_ratio` in the case where the total tokens in the SIC are zero. It returns zero though in the case where the token balance of the SIC is positive, but the total shares are zero in line `84`. That could happen if all deposits have been redeemed, but the SIC still owns a remainder, e. g. from integer division or unclaimed rewards. In such a case, the next user to deposit tokens will not get any shares assigned in `contracts/scc/src/contract.rs:1162`, and hence lose their deposit.

Recommendation

We recommend returning the `default_s_t_ratio` in the case where `total_strategy_shares` are zero in `contracts/scc/src/helpers.rs:82`.

Status: Resolved

5. Multiple reward coins sent to SIC auto-compound and base contracts or coins sent in wrong denom will be locked inaccessibly in contract

Severity: Critical

The check for multiple coins sent in `contracts/sic-auto-compound/src/contract.rs:510` and `packages/sic-base/src/contract.rs:126` as well as the check for the wrong coin denom in lines `516` and `132` of those respective contracts do not return an `Err`, but rather `Ok`, which means that the transactions will not be reverted. Consequently, multiple coins sent or coins sent in the wrong denom will not be returned, but rather locked inaccessibly in the contract.

Recommendation

We recommend returning the `MultipleCoinsSent` or `DenomDoesNotMatchStrategyDenom` errors instead.

Status: Resolved

6. Undelegating from a strategy with not sufficient funds will lead to users losing (part of) their rewards

Severity: Critical

In the `undelegate_from_strategies` function in the SCC contract, an iteration over all strategies is performed, sending the `UndelegateRewards` message for each strategy to its SIC contract. That message specifies the amount to be unbounded in the next undelegation batch in `contracts/scc/src/contract.rs:487`. The handler for that `UndelegateRewards` message in the SIC auto-compound contract checks whether enough funds are available and returns an `Ok` message if funds do not suffice in `contracts/sic-auto-compound/src/contract.rs:571`. Since an `Ok` return value does not revert the transaction, no rewards will be undelegated from the SIC, but the SCC will be updated that the undelegation batch was processed, including updated shares and a new release time for the unbonding. No unbonding happened though, and thus users will not get the funds they should have gotten, with no way to recover.

Recommendation

We recommend returning an `Err` instead of an `Ok` value from the `UndelegateRewards` message handler in the case of not sufficient funds. The SCC contract should then only apply the state changes for the strategy in a sub-message reply if the message was successful.

Status: Resolved

7. Replacing a validator that is receiving a redelegation will lead to loss of delegation

Severity: Critical

When replacing a validator in the SIC auto-compound contract, only the funds in the `can_redelegate` field will be redelegated in `contracts/sic-auto-compound/src/contract.rs:310`. The `can_redelegate` field will be zero if the outgoing validator is currently receiving a redelegation, and contain the full delegation amount otherwise. In the case where the outgoing validator receives a redelegation, no funds will be redelegated, but the validator will still be removed from the `validator_pool` list in line `337`. The delegation amount and the received redelegation will be locked in the pool due to the condition in line `268`.

Recommendation

We recommend returning an error if the validator is currently receiving a redelegation.

Status: Resolved

8. Usage of stored rather than queried delegation amounts may lead to an inconsistent state when unbonding

Severity: Major

The current logic that processes undelegation batches in the pool contract iterates over a pool's validators in `contracts/pools/src/contract.rs:413` to undelegate enough stake to satisfy the amount requested from users. That logic uses the stored stake of validators, rather than querying the currently delegated funds. This is problematic, since validators might have been slashed/jailed or decided to unbond since the storage was last updated. In those cases, the actual undelegated amount would be smaller than the amount requested by users, since Cosmos SDK caps the undelegated amount to the available amount, rather than returning an error. The pools contract would not be aware of the discrepancy, and the stored stake would be inconsistent with actual delegations.

Recommendation

We recommend using actually delegated amounts when undelegating, rather than relying on stored amounts for storage consistency.

Status: Resolved

9. Util functions iterating over unordered hash maps may lead to Terra node consensus failures

Severity: Major

The `map_to_coin_vec` and `map_to_deccoin_vec` functions iterate over the keys of a `HashMap` in `packages/stader-utils/src/coin_utils.rs:111, 121` as well as in `packages/sic-base/src/helpers.rs:184`. Since `HashMap` keys are returned in an arbitrary order, the resulting vector has an arbitrary order of its entries. That could lead to different stored data in the contract storage between Terra nodes, which might cause consensus failures.

Recommendation

We recommend ordering the `HashMap` keys before iterating over them.

Status: Resolved

10. Reward swap and transfer does not consider spread, which leads to distribution of slashing insurance funds as rewards

Severity: Major

The validator contract's `swap_and_transfer` function calculates the amount swapped in `contracts/validator/src/contract.rs:648` without considering the spread/fees applied to the swap. The swap on Terra is at least 2%, but could be higher. Consequently, transferred rewards are higher than they are supposed to be, which leads to funds reserved for slashing being distributed as rewards.

Recommendation

We recommend either querying the swapped tokens from Terra or calculating the received tokens through querying the balances before and after the swap to account for the spread.

Status: Resolved

11. Mismatch of expected and received airdrop amount can lead to lower airdrops for last users withdrawing them

Severity: Major

The `redeem_airdrop_and_transfer` function in `contracts/validator/src/contract.rs:538` as well as the `claim_airdrops` function in `contracts/scc/src/contract.rs:712`, `packages/sic-base/src/contract.rs:71` and `contracts/sic-auto-compound/src/contract.rs:389` expect an input amount that matches the amount received from the airdrop. If the actual amount received from the airdrop is different, accounting of fund allocation in the contract will be off, which might lead the last users to withdraw receiving less than their staked amount.

Recommendation

We recommend querying the amount of received tokens, rather than expecting the caller to specify the correct amount. That can be accomplished by querying and storing the contract's balance of the expected token, then making the claim call, and then querying and comparing the updated balance.

Status: Resolved

12. Detached airdrop pointers update might lead to users receiving less airdrops than expected

Severity: Major

The `update_airdrop_pointers` function in `contracts/pools/src/contract.rs:533` is detached from the actual receipt of the airdrop. If the `airdrop_amount` supplied to the function is different from the received airdrop amount, accounting of funds will be off. The same issue exists in the `update_user_airdrops` function in `contracts/scc/src/contract.rs:1215`. Here the pointers are updated without sending the tokens at the same time.

This issue could lead to the last users trying to unstake not receiving their whole stake back.

Recommendation

We recommend removing the `UpdateAirdropPointers` message from the pools contract, and instead updating the pointers atomically after receiving the funds to make sure they are correctly accounted for. For the SCC contract, we recommend sending airdropped tokens and running the update user aidrops logic in a CW20 receive message handler.

Status: Resolved

13. Discretionary reward and airdrop allocation in delegator contract might lead to incorrect accounting of rewards and airdrops and lower amount of rewards available for withdrawals

Severity: Major

The `allocate_rewards_and_airdrops` function in `contracts/delegator/src/contract.rs:377` allows the manager to allocate additional rewards and airdrops outside of the built-in features of the protocol. There is no validation that those allocated rewards and airdrops have actually been sent to the delegator contract. Again, this might cause a wrong accounting of funds, leaving delegators unable to unstake all of their funds.

Recommendation

We recommend removing the `AllocateRewards` message, and instead handle all reward and airdrop allocations within the protocol. A permissionless claim function that queries the pointers from the pools contract is a less centralized alternative.

Status: Resolved

14. Rewards sent to a strategy are lost if strategy's get total tokens query returns an error

Severity: Major

If the `get_strategy_shares_per_token_ratio` function called in the `try_deposit_funds_to_strategies` function in `contracts/scc/src/contract.rs:1112` returns an error, the strategy will be skipped in line `1118` and the deposited tokens dedicated to that strategy will be stuck in the contract, instead of assigned to the user. The `get_strategy_shares_per_token_ratio` function will return an error if the SIC's `GetTotalTokens` query fails.

We only classify this issue as major since an error during an SIC's `GetTotalTokens` message is unlikely to happen.

Recommendation

We recommend returning an error in line `contracts/scc/src/contract.rs:1118` or sending the funds to the default or retain strategy instead.

Status: Resolved

15. Compromised SCC manager keys could prevent undelegation and fetching undelegated rewards from strategies

Severity: Major

Currently, `UndelegateFromStrategies` and `FetchUndelegatedRewardsFromStrategies` in `contracts/scc/src/contract.rs:397` and `258` are permissioned messages, only executable by the SCC manager. That implies that the SSC manager becomes a single point of failure. If control over the SCC manager account is lost, undelegation and fetching of undelegated rewards will not be possible anymore. Also, in theory, the SCC manager could censor individual users.

Recommendation

We recommend allowing permissionless execution of both `UndelegateFromStrategies` and `FetchUndelegatedRewardsFromStrategies` messages.

Status: Resolved

16. Undelegation from SIC auto-compound contract may fail after slashing

Severity: Major

In `contracts/sic-auto-compound/src/contract.rs:606`, rewards are undelegated from the SIC auto-compound contract according to the stored `stake_fraction`. That `stake_fraction` may be outdated though, since slashing may have happened since the last `stake_fraction` update in the contract. In such a case, the amount to be unbonded could be bigger than the remaining delegation, which would lead to an error that reverts the whole unbonding request.

Additionally, the `stake_fraction` is not re-calculated for all validators every time the total staked amount changes, such that the sum of all stake fractions will likely not be 1.

Lastly, when removing or replacing a validator, the new `stake_fraction` will be calculated in `contracts/sic-auto-compound/src/contract.rs:228` and `317` using the currently delegated amount, while the total staked tokens amount is read from storage. If slashing occurred, the stake fraction of the validator would be too low, so again, the sum of all stake fractions would not equal 1.

We classify this issue as only major since the `stake_fraction` can be updated by removing or replacing all affected validators.

Recommendation

We recommend removing the `stake_fraction`, since it is likely outdated right after it was calculated. Instead, the amount to be unbonded from a validator should be determined by querying current delegations. We also recommend using undelegations to even out differences in stake for a better diversification.

Status: Resolved

17. Swapping contracts with new ones without state migrations may cause problems with funds withdrawal

Severity: Major

In `contracts/validator/src/contract.rs:863-868`, `contracts/pools/src/contract.rs:596-601`, `contracts/delegator/src/contract.rs:484-487`, `contracts/scc/src/contract.rs:192-194`, and `contracts/sic-auto-compound/src/contract.rs:126-128`, contract addresses stored in the config can be updated by the manager. Since state is shared across those contracts, any such upgrades would require either an empty/default state, or a state migration to not cause inconsistencies. Any divergence in state that is shared would lead to severe bugs in the protocol.

Recommendation

We recommend removing the ability to update contract addresses.

Status: Resolved

18. Missing tax deduction may lead to usage of user funds, slashing compensation reserves or rewards for taxes

Severity: Minor

In multiple places in the codebase, taxes are not deducted from the sent native tokens. Taxes are currently charged on any native token other than LUNA, which is tax-free. The Stader contracts can be initialized with any native token denom. If initialized with any token other than LUNA or if LUNA will ever be liable to taxes in the future, the contract's funds will be spent to pay for taxes. The places are:

- `packages/stader-utils/src/helpers.rs:6-13`
- `contracts/scc/src/contract.rs:910`
- `contracts/scc/src/contract.rs:1191`
- `packages/sic-base/src/contract.rs:190`

This has various consequences, for example:

On fund withdrawal in the delegator contract, the transfers in `contracts/delegator/src/contract.rs:364` and `370` will consume not yet withdrawn undelegated user funds. That will leave the last user unable to withdraw their stake.

Withdrawal of pending rewards in the SCC contract in `contracts/scc/src/contract.rs:567` and `910` as well as depositing funds to strategies in line `1191` will likely fail since the contract should not hold any native token balance.

On transfer of undelegated rewards from the auto compound strategy contract in `contracts/sic-auto-compound/src/contract.rs:673`, funds reserved for slashing compensation, as well as uninvested rewards, will be consumed.

In the cases of swap and transfer, transfer of reconciled funds, and removal of slashing funds in the validator contract in `contracts/validator/src/contract.rs:717`, `777` and `841`, funds dedicated to compensate slashing will be used.

We classify this issue as minor instead of critical since it can be recovered from by anyone by sending the relatively small tax amount manually to the contract.

Recommendation

We recommend deducting taxes before transferring tokens. A reference implementation can be found in TerraSwap's code.

Status: Resolved

19. If Terra were to change the 21 day unbonding period, Stader would run into slashing accounting errors if Stader continues using the 21 day unbonding period

Severity: Minor

The pools and SCC contracts have a configurable `unbonding_period`, which is used to determine whether unbonded tokens can be withdrawn. If the value is set lower than the underlying blockchain's unbonding period, several issues arise:

The validator contract may send back undelegated funds from other pools if the `unaccounted_base_funds` is big enough, see `contracts/validator/src/contract.rs:767`. If they are not big enough but enough slashing coverage is available, that slashing coverage will be consumed, see line `772`. In both cases funds are spent in unintended ways. Only if the slashing coverage is also not sufficient, the call will fail in line `770`.

The SCC contract will consider the difference between the expected undelegated funds and the actual available funds as slashed and store it in the `unbonding_slashing_ratio` in `contracts/scc/src/contract.rs:354`. That slashing will later be applied to users that withdraw their rewards, and users will receive fewer rewards than they earned.

Recommendation

We recommend querying the unbonding period from the underlying blockchain, rather than storing it in the config.

Status: Acknowledged

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

20. Validator stake state duplication is error-prone and inefficient

Severity: Minor

Currently, a validator's stake is stored twice, once in the pool contract's `VALIDATOR_REGISTRY` in `contracts/pools/src/state.rs:70` and once in the validator contract's `VALIDATOR_REGISTRY` in `contracts/validator/src/state.rs:28`.

State duplication like that is prone to errors and leads to increased gas consumption.

Recommendation

We recommend only storing a validator's stake in the validator contract and querying it from the pools contract.

Status: Resolved

21. Validator assignment for new deposits does not account for rewards and slashing

Severity: Minor

During the deposit flow in the pool contract, the `get_validator_for_deposit` function is called to pick the validator that will receive the deposit as a bond. Within that function, the validator with the smallest stored stake will be chosen in `contracts/pools/src/request_validation.rs:L90`. Using the stored stake here does not account for rewards and potential slashing that happened since the last storage update. That means that a validator might be chosen that is currently jailed or a validator may be chosen that has not the smallest delegation.

Recommendation

We recommend querying the current delegation and using it to pick the validator rather than using the stored staked amount.

Status: Resolved

22. Contract lookup by address in stader hub might lead to callers running out of gas

Severity: Minor

When looking up a contract by its address in the Stader hub in `contracts/stader-hub/src/contract.rs:94`, a prefix storage query is used to iterate through the stored contracts until a match is found. That is inefficient and indeterministic, since the gas cost varies with the number of contracts stored.

We only classify this issue as minor since the stader hub is currently unused and the expected amount of contracts stored is small, but this issue might cause problems in the future.

Recommendation

We recommend storing a separate map from address to contract name for efficient lookups.

Status: Resolved

23. Lack of address validation might cause errors when using invalid stored addresses

Severity: Minor

In several places in the codebase, the `Addr` type is used for user input in the form of unvalidated addresses. That leaves those addresses unvalidated, which potentially leads to errors later when using an invalid stored address.

Recommendation

It is best practice in CosmWasm to accept addresses as `String` types, and then use `let user_addr: Addr = deps.api.addr_validate(input)?` to validate the address and convert the `String` into an `Addr` type.

Status: Acknowledged

24. Protocol fee values of more than 100% might cause panics

Severity: Minor

The `protocol_fee` config value set in `contracts/delegator/src/contract.rs:37` and `488` is not validated. If it is set to a value greater than 1, `contracts/delegator/src/contract.rs:361` will panic.

Recommendation

We recommend validating the `protocol_fee` to be less than or equal to 1.

Status: Resolved

25. Compromised manager account can take possession of user funds through airdrop claim functions which accept arbitrary messages from manager

Severity: Minor

The airdrop claim functions in `contracts/scc/src/contract.rs:712`, `contracts/sic-auto-compound/src/contract.rs:389`, `contracts/validator/src/contract.rs:544`, and `packages/sic-base/src/contract.rs:79` accept a binary `claim_msg` that will be executed from the respective contracts. While access control is present on these contracts, the caller will be able to pass any call, including CW20 transfers and unbond messages, for example. That could become a potential security issue if the contract holds tokens, is a delegator or has access to trigger permissioned calls on other contracts.

Recommendation

We recommend removing support for arbitrary messages, or moving airdrop claims to a separate contract that neither holds tokens, nor has any other permissions for a cleaner separation of concerns.

Status: Resolved

26. Unbounded iteration over (reward) undelegation batches could run out of gas, leaving funds inaccessible

Severity: Minor

The `reconcile_funds` function of the pools contract as well as the `fetch_undelegated_rewards_from_strategies` function of the SCC contract contain unbounded loops over undelegation batches in `contracts/pools/src/contract.rs:462` and in `contracts/scc/src/contract.rs:292`. These loops could run out of gas, leaving undelegated funds and rewards inaccessible.

While this issue could have fatal consequences, the likelihood of so many not yet reconciled undelegation batches is extremely low, so we consider this issue only as minor.

Recommendation

We recommend adding a limit field to the `ReconcileFunds` and `FetchUndelegatedRewardsFromStrategies` messages to give the manager an option to restrict the amount of iterations and recover from such situations.

Status: Resolved

27. Multiple entries in pool pointers in delegator contract's reward and airdrops allocation function might lead to overwrites

Severity: Minor

In `contracts/delegator/src/contract.rs:394`, a duplicate `pool_id` in the `pool_pointers` vector could lead to overwrites of previous pool pointers.

Recommendation

We recommend asserting that the return value of the `insert` function in line `394` is `None` to ensure that no value has been overwritten.

Status: Resolved

28. Transferring undelegated rewards from the SIC base does not update state

Severity: Minor

The `transfer_undelegated_rewards` function in `packages/sic-base/src/contract.rs:174` does not update the `STATE` to reduce `total_rewards_accumulated` by the transferred amount. There is also no error currently if the amount in state is insufficient.

We only classify this issue as minor since the SIC base contract is just a reference implementation, but it might lead to problems if it is used in new strategies in the future.

Recommendation

We recommend adding validation of the returned amount and updating the state accordingly.

Status: Resolved

29. The check equal coin vector util function fails to detect differences in vectors with duplicates

Severity: Minor

The `check_equal_coin_vector` function in `packages/stader-utils/src/coin_utils.rs:102` does not contain a check for duplicates, so `["1a", "1a", "2b"]` would equal `["1a", "2b", "2b"]`.

Recommendation

We recommend adding a check for duplicates.

Status: Acknowledged

Stader removed usage of this function from production code. The function is now only used in tests, so Stader decided not add a check for duplicates.

30. Lack of validating of default user portfolio may lead to users receiving free tokens

Severity: Minor

The default user portfolio set in `contracts/scc/src/contract.rs:49` is not validated. If the sum of deposit fractions is greater than 100, users using that default portfolio might get free tokens through the strategy allocation in `contracts/scc/src/helpers.rs:137`. That misallocation of funds will leave other users receiving less rewards than earned.

Recommendation

We recommend applying the `validate_user_portfolio` function to the `msg.default_user_portfolio` prior to `contracts/scc/src/contract.rs:49`.

Status: Resolved

31. Lack of validation of default fallback strategy may lead to rewards being stuck in SCC contract

Severity: Minor

The default fallback strategy set in `contracts/scc/src/contract.rs:50` is not validated to exist. If it does not exist, it may still be used in `contracts/scc/src/helpers.rs:144` and `172`, but would lead to an entry in the `failed_strategies` vector in `contracts/scc/src/contract.rs:1103`. That would leave funds stuck in the SCC contract.

Recommendation

We recommend validating that the default fallback strategy actually exists in the instantiate function, or alternatively removing the ability to set a default fallback strategy during instantiation.

Status: Resolved

32. Lack of check that fallback strategy is active may lead to rewards assigned to inactive fallback strategy

Severity: Minor

The `get_expected_strategy_or_default` function in `contracts/scc/src/helpers.rs:103` and `106` does currently not validate that the fallback strategy is active, which could lead to funds being assigned to an inactive fallback strategy.

Recommendation

We recommend adding a check whether the fallback strategy is active.

Status: Resolved

33. Unbounded undelegation records may lead to loss of user funds

Severity: Minor

Undelegation records for users are stored in a vector, which could potentially grow too big to be processed in `contracts/scc/src/contract.rs:864` within a single message, leading to a loss of funds for a user. An example could be a bot that undelegates automatically without fetching the unbonded rewards.

Recommendation

We recommend storing undelegation records in a map for ranged access or limiting the amount of undelegation records that can be stored per user.

Status: Resolved

34. SIC auto-compound contract does not assert that manager seed funds have been sent, could cause users getting less tokens than bonded when undelegating

Severity: Minor

The `instantiate` function in `contracts/sic-auto-compound/src/contract.rs:26` does not verify that the `manager_seed_funds` have actually been sent to the SIC auto-compound contract. If the amount is different from what has been provided, the calculation of unaccounted funds used for undelegation in `contracts/sic-auto-compound/src/helpers.rs:6` will be off, which could cause users receiving less tokens than owed when unbonding.

Recommendation

We recommend asserting that the seed funds have been sent with the instantiate message.

Status: Resolved

35. SIC auto-compound contract does not swap rewards before reinvesting

Severity: Minor

The `transfer_rewards` function of the SIC auto-compound contract does not swap rewards before reinvesting them in `contracts/sic-auto-compound/src/contract.rs:537`. That leads to tokens not in the staking denom (e. g. from airdrops) not being reinvested.

Recommendation

We recommend swapping rewards before they are reinvested.

Status: Acknowledged

Stader currently only reinvests transferred rewards which do not need to be swapped.

36. Reward reinvestment in SIC auto-compound contract does not exclude jailed validators and does not even out differences in delegations

Severity: Minor

In `contracts/sic-auto-compound/src/contract.rs:738`, uninvested rewards are reinvested evenly across all validators in the validator pool. That has two negative effects:

1. Slashed and jailed validators will receive delegations, although they will not be able to produce any rewards.
2. An uneven distribution is not equalized, leading to a less than optimal diversification across validators.

Recommendation

We recommend excluding jailed validators from re-delegation and evening out unequal delegations during the reinvestment handler.

Status: Resolved

37. Different casing of name and address of retain rewards strategy may break other protocols

Severity: Minor

The retain rewards strategy name is an all upper case `"RETAIN_REWARDS"` in the instantiate function in `contracts/scc/src/contract.rs:72`, but an all lower case `"retain_rewards"` in the response to the `GetAllStrategies` query in `contracts/scc/src/contract.rs:1372`. Likewise, the default `StrategyInfo` has its `sic_contract_address` set to `Addr::unchecked("default-sic")` in `contracts/scc/src/state.rs:90`, while the address is set to `Addr::unchecked("")` in `contracts/scc/src/contract.rs:1380`. These differences could potentially break integrations with other protocols.

Recommendation

We recommend putting the name and address into a constant to ensure consistency throughout the codebase.

Status: Resolved

38. Stader bonds new deposits to validator with smallest stake and unbonds from validator with biggest stake, rather than evenly distributing the stake across a pool, leading to lower diversification

Severity: Informational

The current implementation does not equalize the amount staked across all validators on every user deposit/unbond, but rather bonds to the validator in the pool with the smallest stake in `contracts/pools/src/contract.rs:231` and unbonds from the validator in the pool with the biggest stake in `contracts/pools/src/contract.rs:416`. While potentially more gas efficient, this will lead to a less even state distribution, increasing the severity of certain slashes.

Recommendation

We recommend equalizing the stake when depositing and unbonding for a stronger risk diversification.

Status: Acknowledged

The Stader team chose this approach to optimize for gas. The manager can re-delegate to equalize stakes at any time.

39. Users are subject to slashing between reward unbonding and fetching of unbonded rewards

Severity: Informational

The SCC contract allows undelegations at any time, but slashing is effectively only considered when the undelegated rewards are fetched in `contracts/scc/src/contract.rs:343`. This implies that users are subject to slashing long after they unbond. 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 unbonded rewards in Stader will still be subject to slashing until the undelegation is fetched.

Status: Acknowledged

40. Updating user portfolios in SCC contract does not trigger re-allocation

Severity: Informational

The SCC contract's `UpdateUserPortfolio` message allows users to update their portfolios. It does not trigger any re-allocation of currently deployed rewards though, which might be misleading to users.

Recommendation

We recommend re-allocating the rewards according to the updated portfolio.

Status: Acknowledged

The Stader team describes this as intended behaviour – the portfolio will only be updated in the future.

41. Stader hub's query function is unbounded

Severity: Informational

The `query_contracts` function in the Stader hub contains an unbounded storage access in `contracts/stader-hub/src/contract.rs:134` that reads all stored contracts, which might cause calling contracts to run out of gas.

Recommendation

We recommend adding pagination.

Status: Resolved

42. Contracts store addresses to other contracts explicitly instead of using Stader hub

Severity: Informational

The Stader hub contract is a central registry for contract address lookups, but is not currently used by any of the contracts.

Recommendation

We recommend either removing the Stader hub, or using it for address lookups.

Status: Acknowledged

43. Choosing validators to undelegate from is inefficient

Severity: Informational

In the pool contract's undelegation function, a loop is used in `contracts/pools/src/contract.rs:415` to select validators to undelegate from. That loop continues until the amount to be undelegated is zero. In each iteration, the validator with the biggest stake in the `validators` vector is selected by reading all validators from the

`VALIDATOR_REGISTRY` storage in the `get_validator_for_undelegate` function. That is inefficient since storage reads are relatively gas expensive.

Recommendation

We recommend reading all stored validators once outside of the loop to increase the efficiency of the loop.

Status: Resolved

44. Overflow checks not set for release profile in packages/stader-utils/Cargo.toml

Severity: Informational

While set in all other packages, `packages/stader-utils/Cargo.toml` does not enable overflow-checks for the release profile. While the workspace `Cargo.toml` implicitly enables overflow-checks for all packages, a future refactor may break that assumption, which could lead to security issues.

Recommendation

We recommend enabling overflow checks in every package, even if no calculations are currently performed in the package to prevent any issues when the code is extended or refactored in the future.

Status: Resolved

45. Validator contract's redelegate function grants call permission to pools contract, which never calls redelegate

Severity: Informational

The redelegate function in `contracts/validator/src/contract.rs:374` restricts access to the manager, the pools contract or the validator contract. The pools contract currently never calls that function though.

Recommendation

We recommend removing the pools contract from the allowed callers.

Status: Resolved

46. Separate exchange rate queries are inefficient

Severity: Informational

The `query_exchange_rates` function in `packages/stader-utils/src/helpers.rs:17` queries the exchange rate for every passed denom in a separate query call, which is inefficient.

Recommendation

We recommend querying all exchange rates in a single query.

Status: Acknowledged

47. Validator removal reply uses magic number as ID

Severity: Informational

The reply in `contracts/validator/src/contract.rs:915` uses the magic number `0` as the reply ID, while a constant `EVENT_REDELEGATE_ID` exists in the code base.

Recommendation

We recommend using the `EVENT_REDELEGATE_ID` constant.

Status: Resolved

48. Share to token ratio uses magic number as default value

Severity: Informational

The default share to token ratio is set to `Decimal::from_ratio(10_u128, 1_u128)` as a magic number in multiple places, which could negatively impact maintainability and lead to subtle bugs in the future. Instances are:

- contracts/scc/src/contract.rs:631
- contracts/scc/src/contract.rs:1331
- contracts/scc/src/contract.rs:1396
- contracts/scc/src/helpers.rs:71
- contracts/scc/src/state.rs:83
- contracts/scc/src/state.rs:100

Recommendation

We recommend defining a single constant for those values and importing that constant where needed.

Status: Resolved

49. Unused code impacts maintainability

Severity: Informational

The following code is unused and impacts maintainability negatively:

- The `query_airdrop_meta` function in `contracts/pools/src/contract.rs:659`.
- The `redelegations` field of the `UserPoolInfo` struct in `contracts/delegator/src/state.rs:57`.
- The `redelegate` function in `contracts/delegator/src/contract.rs:196` contains commented out code and is not functional.
- The `shares_per_token_ratio` field of the `StrategyInfo` struct in `contracts/scc/src/state.rs:61`. This struct is stored, but the `shares_per_token_ratio` value is re-computed every time it is needed.
- The `released` field of the `BatchUndelegationRecord` struct in `contracts/scc/src/state.rs:215`. This value is not needed, since an `undelegation_batch_status` value of `UndelegationBatchStatus::Done` is equivalent.
- The `get_strategy_apr` function in `contracts/scc/src/helpers.rs:18`.

Recommendation

We recommend removing unused code.

Status: Resolved

50. Duplicated code impacts maintainability

Severity: Informational

All code in `packages/sic-base/src/helpers.rs` is a duplicate of `packages/stader-utils/src/coin_utils.rs`. Duplicated code is harder to maintain.

Recommendation

We recommend removing duplicated code.

Status: Resolved