Security Audit Report

StakingAgency liquid-staking (4)

smart contract

by ARDA

on July 5, 2024

Table of Content

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Inherent risk: A risk for users that comes from a behavior inherent to the smart contract design.

Inherent risks only represent the risks inherent to the smart contract design, which are a subset of all the possible risks. **No inherent risk doesn't mean no risk.** It only means that no risk inherent to the smart contract design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the smart contracts deployed as upgradeable also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Audit Summary

Scope of initial audit

- **Repository:** https://github.com/stakingagency/salsa-sc
- **Commit:** `7bb4bb407228c11f80eb797ff137c6c5be33982e`
- **Path to Smart contract:** `./`

Scope of final audit

- **Repository:** https://github.com/stakingagency/salsa-sc
- **Commit:** `b0f7cd19a100d35752c2a07aac73d02165c482d3`
- **Path to Smart contract:** `./`

Report objectives

1. Reporting all **inherent risks** of the smart contract.
2. Reporting all **issues** in the smart contract **code**.
3. Reporting all **issues** in the smart contract **test**.
4. Reporting all **issues** in the **other** parts of the smart contract.
5. Proposing **recommendations** to address all issues reported.

0 inherent risk in the final commit

4 issues in the final commit

25 issues reported from the initial commit and 4 remaining in the final commit:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	5	0	0	1	0	0
Medium	11	0	0	3	0	0
Minor	9	0	0	0	0	0

Code Issues & Recommendations

Since the smart contract code is not open-source, only the remaining issues are published.

C4: EGLD (un)delegated might be much smaller than EGLD waiting for (un)delegation

Severity: Major **Status:** Won't fix

Location

```
service.rs
    get_provider_to_delegate_and_amount
```

Description

Current behavior: The computed delegation amount can be significantly smaller than the EGLDs waiting to be delegated. Similarly for undelegations. This is because the method `get_provider_to_delegate_and_amount` computes the amount to delegate or undelegate based on the difference of top-ups between providers. If top-ups are very close to each other, then the amount to delegate or undelegate will be small even if there is a big amount of EGLD waiting to be delegated or undelegated.

Expected behavior: The EGLD amount being delegated or undelegated should not be too small compared to the amount pending to be delegated or undelegated, otherwise this would make the protocol inefficient, as many transactions will be needed in order to delegate and undelegate all amounts, inducing delays. Moreover, between subsequent delegations and undelegations, providers must be refreshed, which takes at least 3 more blocks, inducing further delays. For delegations, a timelock of 10 blocks must also be respected between delegations.

Worst consequence: It could take up to a week to delegate all deposited users' EGLD, during which these EGLD do not earn rewards.

Example: Let's suppose that there are 2 providers with 1 staked node each, that their top-up difference is 0.5 EGLD, and that there are 10,000 EGLD

waiting to be delegated. In such a scenario, the EGLD sent in the transaction will be 1 EGLD. Since providers are selected alternately, the top-up difference will remain unchanged. Therefore, the contract will have to send 10,000 transactions. Since delegations must be spaced by at least 10 blocks, i.e. 1 minute, the minimum duration needed for the contract to delegate all the 10,000 EGLD is 6 days, 22 hours, and 40 minutes.

A similar example can be done for undelegation, with the difference that we do not have to wait for 10 blocks between each undelegation transaction, but rather 3 blocks in order to refresh providers between consecutive undelegation transactions.

Recommendation

At a high-level, the solution recommended below tries to delegate (resp. undelegate) `min(amount, max(5% amount, 50))` from the provider with min top-up (resp. max top-up).

This solution ensures that in 1 hour more than 95% of the pending amount will be delegated or undelegated. In the example above, it would have taken ~1 hour to undelegate 10,000 EGLD compared to ~7 days of the current implementation.

Note that this solution doesn't use `dif_topup` in the calculation of the amount to delegate or undelegate, hence:

- It helps converging faster to the equalization of top-ups. This is because (un)delegating the amount `dif_topup * staked_nodes` pushes the top-up of the provider directly to the other extreme, while the proposed solution pushes it more moderately towards the average.
- It significantly simplifies the algorithm. Hence any computation related to `dif_topup` can be removed from the codebase.

For delegation

We start by defining `amount_to_delegate = min(amount, max(5% amount, 50))`. Then:

1. We try to find the active elligible provider with the smallest top-up among all active elligible providers that also have a free space of at least `amount_to_delegate`. A function `find_active_elligible_provider_min_topup_to_delegate` can be

introduced for this, returning the provider and its top-up. If such a provider is found, we delegate `amount_to_delegate`.

2. Otherwise, we try to find the active elligible provider with the highest available space (which may be infinite). A function `find_active_elligible_provider_with_highest_space_to_delegate` can be created to do this. If such a provider is found, we delegate the amount `min(amount_to_delegate, provider.max_cap - provider.total_stake)`.

For undelegation

All the steps below for selecting a provider from which to undelegate will need the common function `compute_amount_to_undelegate` suggested in Can choose an amount that can't be undelegated from a provider .

We start by defining `initial_amount_to_undelegate = min(amount, max(5% amount, 50))`. Then:

1. We try to find an inactive or uneligible provider such that `compute_amount_to_undelegate(initial_amount_to_undelegate, salsa_stake) > 0`. A function `find_inactive_or_uneligible_provider_to_undelegate` can be introduced for this. If such a provider is found, we undelegate the amount `compute_amount_to_undelegate(initial_amount_to_undelegate, salsa_stake)` and we stop here.
2. Otherwise, we try to find the provider with the maximum top-up among providers such that `compute_amount_to_undelegate(initial_amount_to_undelegate, salsa_stake) > 0`. A function `find_provider_max_topup_to_undelegate` can be introduced for this, returning the provider and its top-up. If such a provider is found, we undelegate the amount `compute_amount_to_undelegate(initial_amount_to_undelegate, salsa_stake)` and we stop here.
3. Otherwise, we try to find the provider with the highest Salsa stake such that `compute_amount_to_undelegate(initial_amount_to_undelegate, salsa_stake) > 0`. A function `find_provider_max_salsa_stake_to_undelegate` can be introduced for this. If such a provider is found, we undelegate the amount

`compute_amount_to_undelegate(initial_amount_to_undelegate, salsa_stake)`.

Resolution notes

The issue is not entirely and correctly solved. Below we list the remaining problems.

Remaining issue 1: In `get_provider_to_delegate_and_amount`, if no provider is found with a sufficiently big free space to delegate the initial amount entirely, then no EGLD are delegated, although the recommendation suggests calling a method `find_active_elligible_provider_with_highest_space_to_delegate` to delegate the maximal possible amount.

Therefore, the delegation follows an all-or-nothing principle: either the prescribed amount of EGLD is delegated, or 0 EGLD is delegated.

There is the analogous problem with undelegations, i.e. no EGLD are undelegated if it is not possible to undelegate the prescribed EGLD amount entirely, however it would be possible to undelegate some EGLD, by using the method `find_provider_max_salsa_stake_to_undelegate` from the recommendation.

Recommendation: We can stick to the original recommendation, i.e. introducing the functions `find_active_elligible_provider_with_highest_space_to_delegate` and `find_provider_max_salsa_stake_to_undelegate` and calling them in `get_provider_to_delegate_and_amount` and `get_provider_to_undelegate_and_amount`.

Remaining issue 2: From the recommendation, calculating the variable `dif_topup` is now useless, because we can remove the difference of top-ups from the selection algorithm, to get a simpler and more efficient algorithm.

Recommendation: As described in recommendation, we can remove the logic for computing differences of top-ups, when selecting providers for delegations and undelegations.

Remaining issue 3: The function `compute_amount_to_undelegate` is unnecessarily complicated compared to the suggested function in Can choose an amount that can't be undelegated from a provider , and this complexity might lead to an unintended amount of EGLD to undelegate in some case.

More precisely, `compute_amount_to_undelegate` takes top-ups `min_topup` and `max_topup` as arguments, and in case there difference is 0, `dif_topup = max_topup - min_topup == 0`, it returns that the full amount of EGLD given as argument should be undelegated.

This was implemented because `compute_amount_to_undelegate` is used in two separate contexts: once for the normal selection algorithm where top-ups matter, but also once for uneligible and inactive providers, in which case we don't care about top-ups and hence arguments `max_topup = min_topup = 0` are provided. This is for this reason that in `compute_amount_to_undelegate`, we interpret `dif_topup == 0` as if the selected provider was unelligible or inactive, in which case it is desired to undelegate the initial amount entirely.

However, the side effect of this logic is that, in case of an eligible and active provider selected by the normal procedure, we could have `dif_topup == 0`, and in this case we would undelegate the initial amount entirely from the provider, which is a discontinuous and unexpected behavior.

Example: If the initial amount to undelegate is 1000 EGLD, and the selected provider from which to undelegate has more than 1000 EGLD staked, then we would have the following discontinuous behavior:

- If `dif_topup > 0`, 5% of the initial amount is undelegated: 50 EGLD.
- If `dif_topup == 0`, all initial amount is undelegated: 1000 EGLD.

Recommendation: To solve this, we can follow the recommendation, for one thing by avoiding the `dif_topup` computations as explained in the previous issue, and moreover by keeping `compute_amount_to_undelegate` as simple as possible, to avoid handling several cases (uneligible and eligible providers) in the same function.

C6: Can add as provider an address which is not a provider

Severity: Medium **Status:** Won't fix

Location

```
providers.rs
    add_provider
```

Description

Current behavior: Any address can be added in the list of providers.

Expected behavior: Only the addresses of providers' contracts should be added in the list of providers. Otherwise, if a wrong address is added, then it might be impossible to call the expected endpoints to refresh the delegation data, and in turn users won't be able to undelegate and withdraw their funds from any provider.

Worst consequence: An erroneous provider address is added in the list of providers but the contract's owner is unable to quickly remove the incorrect address e.g. if he lost his keys, is ill or has no access to a computer. Then during this time all users can't withdraw their funds.

Recommendation

In `add_provider`, to ensure that the address is a legitimate provider, we recommend adding the following checks:

- That the address is on the metachain. This can be done by checking `self.blockchain().get_shard_of_address(address) == METACHAIN_SHARD_ID`, where `METACHAIN_SHARD_ID` is the constant `u32::MAX`.
- That the address is the one of a provider, by calling an endpoint that only a provider can have on the metachain, e.g. `getContractConfig`. For this, we can add a new boolean argument `add_provider_in_callback` to the callback `get_contract_config_callback`. Then, in `get_contract_config_callback`, if the asynchronous call was successful and if `add_provider_in_callback == true`, we can insert the address of the provider in the list of providers.

Resolution notes

Remaining issue: The 2nd point of the recommendation has not been followed. Therefore it is possible that by mistake, the owner whitelists an address which is on the metachain, but is not the address of a provider. This would result in failed delegations.

Recommendation: We can follow the 2nd point of the recommendation.

C9: Min top-up for undelegation and max top-up for delegation not computed for all active providers

Severity: Medium **Status:** Won't fix

Location

```
service.rs
    get_provider_to_delegate_and_amount
```

Description

Current behavior: When selecting a provider, the minimum top-up for undelegation and the maximum top-up for delegation are computed only among active providers which have free space.

Expected behavior: Both should be computed among all active providers, regardless of whether they have free space or not. This is what is expected by the project, because:

- The undelegation amount is chosen such that the selected provider's top-up does not go below the minimum top-up between all active providers.
- The delegation amount is chosen such that the selected provider's top-up does not go above the maximum top-up between all active providers.

Worst consequence: Wrong top-up computations can lead to too few EGLD delegated or undelegated.

Example: Let's consider 3 providers, each with 1 node, such that:

- Provider 1 has no free space, and a top-up of 500,
- Provider 2 has free space, and a top-up of 1500,
- Provider 3 has free space and a top-up of 2000.

Then, when undelegating EGLD, `get_provider_to_delegate_and_amount` will choose provider 3 to perform the undelegation. Then, the computed `min_topup` will be 1500, which is the top-up of provider 2. However, it it was correctly computed, `min_topup` should would have been 500, i.e. the top-up of provider 1. Consequently, 2000 - 1500 = 500 EGLD will be undelegated, instead of 2000 - 500 = 1500 EGLD.

Recommendation

We recommend following the recommendation to EGLD (un)delegated might be much smaller than EGLD waiting for (un)delegation, as it resolves this issue as well since there is would be no computation of top-ups difference anymore.

Alternatively, we recommend making the following changes in `get_provider_to_delegate_and_amount`:

- Renaming `max_topup_delegate` into `max_topup`, and updating it no matter if the provider has free space or not.
- Renaming the current variable `min_topup` into `min_topup_delegate`. It is the top-up of the provider selected for delegation.
- Introducing a new `min_topup` variable, which is the minimum top-up among all active providers, and thus updating it no matter if the provider has free space or not.

As a result of these changes, the values `min_topup_delegate` and `max_topup` will be used for determining the delegation amount, while the values `max_topup_undelegate` (already correctly implemented) and `min_topup` will be used for determining the undelegation amount.

Resolution notes

Remaining issue: The recommendation is not entirely followed, because for delegations the maximal top-up is still computed among providers *which have free space*, although it should be computed among all providers.

Recommendation: We suggest following the recommendation. In particular, the main solution consists in following the recommendation to EGLD (un)delegated might be much smaller than EGLD waiting for (un)delegation, which simply removes all the computations of top-ups differences.

C11: get_provider_to_delegate_and_amount is unnecessarily complex

Severity: Medium **Status:** Won't fix

Location

```
service.rs
    get_provider_to_delegate_and_amount
```

Description

Current behavior: The function `get_provider_to_delegate_and_amount` is unnecessarily complex and performs multiple independent logics, one for selecting a provider in which to delegate and one for selecting a provider from which to undelegate.

Expected behavior: The logic for selecting providers in which to delegate and from which to undelegate should be separated, as they are independent. Moreover, each of these logics should be made as simple as possible. This is because complex functions increase the risks of introducing errors and any error there could result in wrong funds allocation or even losses of funds.

Recommendation

We recommend to replace `get_provider_to_delegate_and_amount` with two functions: `get_provider_to_delegate_and_amount`, for delegation, and `get_provider_to_undelegate_and_amount`, for undelegation.

Each of these functions can then be further broken down into smaller, more manageable parts using helper functions. These helpers functions are those detailed in the recommendation of EGLD (un)delegated might be much smaller than EGLD waiting for (un)delegation.

Resolution notes

Remaining issue: Although the selection of providers for delegations and undelegations has been separated in two methods `get_provider_to_delegate_and_amount` and `get_provider_to_undelegate_and_amount` as recommended, these methods

are still highly complex and, unlike the suggestion from the recommendation, they have not be broken down into smaller and simpler functions.

Recommendation: It would be good to aim for simpler functions, following the break down into smaller functions from the recommendation of EGLD (un)delegated might be much smaller than EGLD waiting for (un)delegation.