Security Audit Report

Hatom liquid-staking (2)
MultiversX smart contract

by ARDA

on October 15, 2025

Table of Contents

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

Code: The code with which users interact.

Inherent risk: A risk for users that comes from a behavior inherent to the code's design.

Inherent risks only represent the risks inherent to the code's 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 code's 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 upgradability of the code might 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.

Objective

Our objective is to share everything we have found that would help assessing and improving the safety of the code:

1. The **inherent risks** of the code, labelled R1, R2, etc.
2. The **issues** in the **code**, labelled C1, C2, etc.
3. The **issues** in the **testing** of the code, labelled T1, T2, etc.
4. The **issues** in the **other** parts related to the code, labelled O1, O2, etc.
5. The **recommendations** to address each issue.

Audit Summary

Initial scope

- **Repository:** https://github.com/HatomProtocol/hatom-liquid-staking
- **Commit:** `6818e1ea5284d699a1a1dd9a3449de04f88d2df4`
- **MultiversX smart contract path:** ./liquid-staking/

Final scope

- **Repository:** https://github.com/HatomProtocol/hatom-liquid-staking
- **Commit:** `e367ad690f0dd0ad84068227ed8a9e4e89e3002c`
- **MultiversX smart contract path:** ./liquid-staking/

2 inherent risks in the final scope

3 issues in the final scope

12 issues reported in the initial scope and 3 remaining in the final scope:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	2	0	0	0	0	0
Medium	7	1	0	3	0	0
Minor	2	0	0	0	0	0

Inherent Risks

R1: Users might make smaller profits than if they delegated directly to the provider of their choice.

This is because:

1) A protocol fee is applied on the rewards generated from staked EGLD.

2) Users' EGLD might not all be delegated in the providers generating the highest profits:

- Hatom admins decide which providers are allowed for delegations.
- The selection of the provider where users' EGLD are delegated depends on off-chain data about providers provided by an oracle, therefore these data might be inaccurate and/or not favor the providers that would lead to highest profits.
- The algorithm selecting the provider where users' EGLD are delegated / undelegated might not select the providers which generate the highest / lowest profits.
- Only admins can execute the algorithm selecting the provider where users' EGLD are delegated. Therefore if admins are inactive, these EGLD would not generate any profits to users.
- The admins are allowed to undelegate EGLD from a provider of their choice, in order to re-delegate these EGLD afterwards in another provider, chosen by the selection algorithm. Until they are re-delegated, which takes at least 10 days to pass the unbonding period, these EGLD would not generate any profits to users.

R2: Users might not be able to withdraw if their SEGLD are worth less than 1 EGLD.

This is because the undelegation endpoint of the Liquid Staking smart contract does not accept SEGLD amounts worth less than 1 EGLD.

For example, this could happen when:

- The user deposits less than 1 EGLD.
- Or the user transfers parts of his SEGLD and his remaining SEGLD are worth less than 1 EGLD.

Code Issues & Recommendations

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

C3: Users might be unable to undelegate the last 2 EGLD remaining in a provider

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

Description

Current behavior: A user might be unable to undelegate the last 2 EGLD remaining in a provider. This is because undelegating from the provider is forbidden if it would leave the provider with a non-zero amount smaller than 1 EGLD.

Consequently, if the above situation occurs for all whitelisted providers, i.e. the user's undelegation would leave any of these providers with a non-zero amount smaller than 1 EGLD, then the `undelegate` endpoint would fail to select a provider, and the user can't withdraw his funds.

For example, if there is a single user whose SEGLD are worth 1.99 EGLD, and the Liquid Staking smart contract currently has 1.999 EGLD all delegated in a single provider, then the user can't undelegate because:

- It is impossible to undelegate less than 1 EGLD from the provider,
- It is impossible to undelegate any amount between 1 EGLD and 1.99 EGLD: it would leave the provider with a non-zero amount of EGLD smaller than 1.

Expected behavior: Even when a provider has less than 2 EGLD left, users should be able to undelegate from it, if their SEGLD is worth at least 1 EGLD. This is the behavior they would expect based on their experience with direct staking: a user is always able to fully undelegate his EGLD from a provider, and when he does so, he must be undelegating at least 1 EGLD, because it is a constraint of the provider's smart contract that the amount of EGLD delegated by any user is always either 0 or at least 1 EGLD.

Worst consequence: The Liquid Staking smart contract has 1.999 EGLD delegated in all whitelisted providers, i.e. 31 providers at the time of this audit.

Meanwhile, there are 31 SEGLD holders, each of which has an SEGLD amount worth 1.99 EGLD. Therefore, none of these users can withdraw, which means that a total of 61.69 EGLD can't be withdrawn.

Example: A precise scenario where the issue occurs is provided in the "Test 0" of this reproduction. In this scenario, a user deposits EGLD in the Liquid Staking smart contract, which are delegated to a single provider. Rewards are then claimed, which leads to rounding errors in the SEGLD:EGLD conversion rate. In turn, when the user attempts to undelegate, the EGLD value of his SEGLD is slightly underestimating the amount of EGLD delegated in the provider, and it is therefore impossible to undelegate these EGLD.

Recommendation

At a high-level, we recommend that the admin deposits 1 EGLD in a whitelisted provider, before users are allowed to delegate their EGLD in this provider.

In practice, for each provider, we introduce a new boolean storage `initial_egld_delegated`, and in the selection algorithm for delegation, we accept the provider only if `initial_egld_delegated` is `true`. In order to set `initial_egld_delegated` to `true`, the admin would call an admin endpoint `delegate_initial_egld` that takes a provider address as argument and proceeds as follows:

- It requires that 1 EGLD is received and that `initial_egld_delegated` is `false`.
- It delegates the 1 EGLD to the provider.
- In the callback, if the asynchronous call succeeded, it sets `initial_egld_delegated` to `true` (we don't do any other updates, e.g. we don't increase the storages `cash`, `shares`, `total_delegated` and don't mint any SEGLD for the caller). Otherwise, if the asynchronous call failed, we send back the EGLD.

Moreover, we would remove the following lines from `is_valid_undelegation_contract_relaxed`:

```
let min_egld_to_delegate = BigUint::from(MIN_DELEGATION_AMOUNT);
let amount_left = &contract_data.total_delegated - egld_amount;
if amount_left != BigUint::zero()
   && amount_left < min_egld_to_delegate {
   return false;
}
```

Finally, after the Hatom team will have performed the initial delegation in each provider, the corresponding transactions should be shared with the auditor.

C4: Fee for instantaneous undelegations can be too small

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

Location

```
liquid-staking/src/governance.rs
    set_instantaneous_fee
```

Description

Current behavior: The owner can choose any value for the fee `instantaneous_fee` paid by users during instantaneous undelegations. However if it is too small, e.g. 0%, an attacker could steal staking rewards from users who actively delegate their EGLD, by delegating EGLD before rewards are claimed, and exiting straight after by calling `instantaneous_undelegate`. Indeed, if the profit made from rewards is greater than the fee, the attacker would be incentivized to perform the above steps, and would then have earned EGLD rewards he did not deserve, which should have been distributed to other users.

Expected behavior: The instantaneous fee should be sufficiently big such that the above quick enter-and-withdraw attack would make the attacker lose funds rather than making profits.

Worst consequence: If `instantaneous_fee` is 0%, an attacker could steal staking rewards from other users. At each epoch, he would delegate EGLD, trigger himself the claiming of rewards from every providers (as the endpoint for claiming rewards is public), and exiting straight after by calling `instantaneous_undelegate`.

Recommendation

We recommend adding a check in the endpoint `set_instantaneous_fee` to ensure that the fee is at least 0.03%, i.e. no smaller than a constant `MIN_INSTANTANEOUS_FEE = 3`.

The proposed minimal fee of 0.03% has been chosen conservatively to ensure that a quick enter-and-withdraw attack would make the attacker lose funds rather than making profits. Indeed, assuming a 10% APR from staking (which is

higher than the current APR), the rewards that can be claimed at each epoch generate a profit smaller than 0.03%.

Finally, in `instantaneous_undelegate`, in case the open mode is inactive, we recommend checking that `instantaneous_fee` is set, i.e. that it is non-zero.

C9: SEGLD supply can be very small compared to pending rewards and lead to failures at deposits or losses at withdrawals

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

Description

Current behavior: The total supply of SEGLD `ls_token_supply` can be very small, e.g. 0 or close to 0, while there is a significant amount of pending EGLD rewards to be claimed. However, after rewards are claimed and re-delegated, thereby increasing the EGLD reserve `cash_reserve`, the conversion rate between SEGLD and EGLD would explode.

The consequence of a huge SEGLD:EGLD rate is that it could prevent future users from depositing EGLD amounts below a high threshold (e.g. 100 EGLD), and also induce significant rounding errors when they undelegate, leading to losses of funds, as shown in the example below.

Expected behavior: The conversion rate between SEGLD and EGLD should handle the case where most users have withdrawn their SEGLD but there are still significant amounts of rewards to be claimed and re-delegated, ensuring that even in this case users can deposit 1 EGLD and don't lose non-negligible amounts of EGLD when withdrawing.

Worst consequence: If the total supply of SEGLD becomes very small, e.g. close to 0, while there is a significant amount of pending EGLD rewards to be claimed, an attacker could claim and re-delegate the rewards, in order to inflate the SEGLD:EGLD rate and steal the funds of subsequent users because of rounding errors. This is exemplified below.

Example: After most users have withdrawn their SEGLD, there only remains 1 SEGLD owned by an attacker Alice, while there are 100 EGLD (`10^20` atoms of EGLD) rewards pending to be claimed. Alice then claims and re-delegates the rewards, therefore `cash_reserve = 10^20` and `ls_token_supply = 1`, and 1 atom of SEGLD is now worth `10^20` atoms of EGLD. There are then two types of undesired consequences:

- *Impractical Deposits:* From now on, users can't delegate less than 100 EGLD, as otherwise `delegate` would fail when trying to mint 0 SEGLD.
- *New users lose funds:* A user Bob deposits 190 EGLD and receives only `190 * 1 / 100 = 1` SEGLD due to rounding errors. A user Eve deposits

280 EGLD, and receives only `280 * 2 / 290 = 1` SEGLD. A user Carol deposits 370 EGLD, and receives only `370 * 3 / 570 = 1` SEGLD. Alice then withdraws her 1 SEGLD and receives `940 / 4 = 235` EGLD, i.e. she has stolen 135 EGLD. Then, the other users withdraw: Bob earns 45 EGLD, while Eve loses 45 EGLD and Carol loses 135 EGLD.

Recommendation

At a high-level, we suggest that before the state can be activated, the Hatom admin should perform an initial deposit of 1 EGLD and that keeps the corresponding SEGLD in the smart contract instead of sending it back to the caller. This will ensure that even after all users withdraw and if there are pending rewards to be compounded, the conversion between SEGLD and EGLD will still be reliable.

In more details, we introduce a boolean storage variable `initial_deposit_done` to track whether the initial deposit has been made, and in `set_state_active`, we require that `initial_deposit_done` is true. Then, we have an admin endpoint `initial_deposit` that requires 1 EGLD to be sent, checks that `initial_deposit_done` is false, and sets `initial_deposit_done` to true.

Test Issues & Recommendations

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