Audit Report

IncrementFi Liquid Staking

v1.1

March 6, 2023

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

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 Increment Labs Ltd. to perform a security audit of the Increment Finance Liquid Staking smart contracts.

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/IncrementFi/Liquid-Staking

Commit hash: `cdc3b2e509a9e207cf6b1d3d0ce14fb5a61bb82b`

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

The submitted codebase implements a liquid staking protocol by Increment Finance. The protocol allows users to earn staking rewards without locking Flow tokens or running node software. Users can deposit Flow tokens and receive transferrable stFlow tokens in return.

The contracts in-scope to audit include

- `cadence/contracts/DelegatorManager.cdc`
- `cadence/contracts/LiquidStaking.cdc`
- `cadence/contracts/LiquidStakingConfig.cdc`
- `cadence/contracts/LiquidStakingError.cdc`
- `cadence/contracts/stFlowToken.cdc`

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**.

Note that audits are an important step to improving 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 in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	Most functions are well-documented with clear and concise comments.
Level of documentation	High	Detailed documentation is available at https://docs.increment.fi/protocols/liquid-staking and https://developers.flow.com/nodes/staking.
Test coverage	Low	There were no test cases available. The client mentioned that the liquid staking protocol relies on many underlying contracts, and the testing framework runs on the emulator environment, which is different from the mainnet/testnet. This makes it difficult to simulate the epoch turns or reward payment behavior in the test framework.

Summary of Findings

No	Description	Severity	Status
1	Access nodes are not filtered when approving or setting default node	Minor	Acknowledged
2	Migrate function does not enforce minimum staking amount	Minor	Resolved
3	Minimum staking amount and staking cap are not validated against each other	Minor	Acknowledged
4	Trust dependency on admin keys	Informational	Acknowledged
5	Incorrect event emitted when admin modifies `isMigratingPaused` configuration	Informational	Resolved
6	Supply start index higher than the end index causes `collectDelegatorsOnEpochStart` to perform empty execution	Informational	Resolved
7	`initApprovedNodeIDList` gas consumption can be reduced	Informational	Acknowledged
8	Distributing tokens to the same node operator yield no difference	Informational	Resolved
9	Typographic errors and duplicate comments found in codebase	Informational	Resolved
10	Unused events and properties	Informational	Resolved
11	Best practices for transactions code	Informational	Acknowledged
12	Error messages are returned with proprietary encoding	Informational	Acknowledged

Detailed Findings

1. Access nodes are not filtered when approving or setting default node

Severity: Minor

In `cadence/contracts/DelegatorManager.cdc:957` and `984`, the admin can initialize an approved list of node operators or set a default node to stake using the `initApprovedNodeIDList` and `setDefaultNodeIDToStake` functions. However, no validation ensures the initialized or selected nodes are not access nodes.

Since access nodes cannot be delegated to as seen in `cadence/contracts/standard/emulator/FlowIDTableStaking.cdc:1183-1186`, misconfiguring an access node as the default staking node prevents users from staking `FLOW` tokens and redeeming `FLOW` tokens instantly from the liquid staking contract.

This could happen when the admin calls `setDefaultNodeIDToStake` with an access node as the `nodeID` argument or during the advancing epoch phase in the `filterApprovedNodeListOnEpochStart` function at line `633`.

We consider this a minor issue since only the admin can cause it.

Recommendation

We recommend adding checks to ensure the initialized and selected nodes are not access nodes.

Status: Acknowledged

The client mentioned that the admin would check off-chain when setting the config, and the `filterApprovedNodeListOnEpochStart` function will also check the nodes' feasibility on the chain when a new epoch comes.

2. Migrate function does not enforce minimum staking amount

Severity: Minor

In `cadence/contracts/LiquidStaking.cdc:185`, the `migrate` function does not check whether the delegated tokens are greater than or equal to the minimum staking amount. Unlike the `stake` function's precondition check in line `60`, the validation is not enforced in the `migrate` function.

Recommendation

We recommend ensuring the delegated tokens are larger or equal to `LiquidStakingConfig.minStakingAmount`.

Status: Resolved

The client mentioned that the migrate function is mainly used in the migrate.cdc transaction. Before the migrate function, any unclaimed rewards, new token commits, unstaked tokens will be withdrawn and liquid-staked first. There're cases where delegators have < 0.1 staked `FLOW` but with several unclaimed rewards etc. Instead of failing the migration progress, they want to ensure these delegators can successfully migrate in 1 transaction.

3. Minimum staking amount and staking cap are not validated against each other

Severity: Minor

In `cadence/contracts/LiquidStakingConfig.cdc:100-108`, the admin can set the minimum staking amount and staking cap for the liquid staking protocol. Since the minimum staking amount is expected to be lower than the staking cap and vice versa, misconfiguring the values would prevent users from staking their `FLOW` tokens due to `cadence/contracts/LiquidStaking.cdc:60` and `62`. Ideally, there should be a precondition check to ensure the updated values are higher/lower than the other.

We consider this a minor issue since only the admin can cause it.

Recommendation

We recommend adding precondition checks in `setMinStakingAmount` and `setStakingCap` to ensure the new minimum staking amount is lower than `LiquidStakingConfig.stakingCap` while the new staking cap is higher than `LiquidStakingConfig.minStakingAmount`.

Status: Acknowledged

The client mentioned that these two parameters perform their own functions and are not related.

4. Trust dependency on admin keys

Severity: Informational

In `cadence/contracts/stFlowToken.cdc:121`, the `mintTokens` function's access modifier is set to `access(account)`. Theoretically, the account owner can mint as many `stFlow` tokens as required by deploying a new contract that calls the `mintTokens` function or performing a contract upgrade that modifies the access modifier into `pub` or `access(all)` keywords.

The possibility of this happening is that the admin is malicious or the private key is compromised. With that said, both deploy and update contract actions will emit events that off-chain listeners can fetch, which are easily trackable and monitorable with tools.

Recommendation

This is a note to the readers to understand how the `stFlow` token works. We recommend setting up the account as a multi-signature account to avoid any single malicious party from minting tokens.

Status: Acknowledged

The client mentioned that they would consider revoking the account's private key when the project is stable in the future to seek "complete" decentralization. Additionally, they agree that multisig is better and would consider using it once the underlying system staking contracts are matured and finalized.

5. Incorrect event emitted when admin modifies `isMigratingPaused` configuration

Severity: Informational

In `cadence/contracts/LiquidStakingConfig.cdc:137`, the `ConfigStakingPause` event is emitted when the admin updates the `isMigratingPaused` configuration. This is incorrect, as the `ConfigMigratingPause` event should be emitted instead.

Recommendation

We recommend modifying line `137` to emit the `ConfigMigratingPause` event instead.

Status: Resolved

6. Supply start index higher than the end index causes `collectDelegatorsOnEpochStart` to perform empty execution

Severity: Informational

In `cadence/contracts/DelegatorManager.cdc:503`, the while loop attempts to collect all delegators to the next epoch based on the provided start and end index. If the start index’s value is supplied higher than the end index, the loop will not execute, causing the `collectDelegatorsOnEpochStart` function to collect zero delegators.

Recommendation

We recommend adding a precondition check to ensure the `startIndex` is lower or equal to the `endIndex`.

Status: Resolved

7. `initApprovedNodeIDList` gas consumption can be reduced

Severity: Informational

In `cadence/contracts/DelegatorManager.cdc:962-964`, the for loop attempts to insert the `approvedNodeIDList` dictionary with the `nodeIDs` argument key and value. This causes unnecessary gas consumption as the values can be set directly.

Recommendation

We recommend removing the loop and setting the `approvedNodeIDList` value to `nodeIDs` directly.

```
DelegatorManager.approvedNodeIDList = nodeIDs
```

Status: Acknowledged

The client mentioned that `DelegatorManager.approvedNodeIDList` has been set to const type and cannot be assigned again. Since the node list is small, it can be ignored.

8. Distributing tokens to the same node operator yield no difference

Severity: Informational

In `cadence/contracts/DelegatorManager.cdc:826`, the `transferCommittedTokens` function allows a strategy bot to transfer committed tokens from one delegator to another. Since no validation ensures that `fromNodeID` and `toNodeID` are not the same node operator, the result of the execution would yield no difference.

Recommendation

We recommend adding a precondition check to ensure that `fromNodeID` and `toNodeID` are not the same node operator.

Status: Resolved

9. Typographic errors and duplicate comments found in codebase

Severity: Informational

In several instances of the codebase, typographical errors were found along with duplicate code comments.

- `cadence/contracts/DelegatorManager.cdc:125` contains an additional "*protocol*" word at the end of the sentence
- `cadence/contracts/LiquidStakingConfig.cdc:13` contains a typographical error of "*minimum*"

This affects the readability of the contracts.

Recommendation

We recommend correcting the errors above.

Status: Resolved

10.Unused events and properties

Severity: Informational

In several instances of the codebase, unused events and contract properties can be removed.

- `cadence/contracts/stFlowToken.cdc:41` unused `MinterCreated` event
- `cadence/contracts/stFlowToken.cdc:44` unused `BurnerCreated` event
- `cadence/contracts/stFlowToken.cdc:149` unused `tokenProviderPath` variable.

This affects the readability of the contracts.

Recommendation

We recommend removing the unused properties to improve the overall readability of the code.

Status: Resolved

11. Best practices for transactions code

Severity: Informational

In several instances of the transactions code, the `prepare` phase includes logic that does not interact with the `AuthAccount` objects of signing accounts. As a best practice, only use the `prepare` phase to define and execute the logic that requires access to the `AuthAccount` while moving all other logic to `execute`, `pre` or `post` phases.

Some of the examples from transactions are:

- `cadence/transactions/system/mint_flow.cdc:11` contains logic that can be moved to execute phase
- `cadence/transactions/system/set_approved_list_in_system.cdc:10` interaction with a reference can be moved to execute phase
- `cadence/transactions/user/stake_with_swap.cdc:22` logic can be moved out of prepare phase

Although other phases are optional, it is a best practice to keep the transaction logic in the respective section explicitly.

Recommendation

We recommend distributing the logic to `execute`, `pre` or `post` phases instead of maintaining everything in the `prepare` phase.

Status: Acknowledged

The client mentioned that they don't define any member variables in transactions for code simplicity. However, since the product has already been launched and the transactions have gone through rounds of testing, reconstruction will increase potential risks. Hence, they chose not to make changes.

12. Error messages are returned with proprietary encoding

Severity: Informational

The function `ErrorEncode` in the contract `cadence/contracts/LiquidStakingError.cdc:23` uses a proprietary syntax to encode the error message into `String`. Using a common syntax like JSON can help decode it natively in many applications and removes the need to write custom logic to do the same.

Recommendation

We recommend encoding the error messages in common encoding formats like JSON.

Status: Acknowledged

The client mentioned that all the Increment products have a unified set of error coding formats (including lending, swap, farming, etc.), and there is a front-end library to identify errors' encoding.