Tokenlabs

Audit Report

contact@bitslab.xyz

https://twitter.com/movebit_

Wed Jan 14 2026

Tokenlabs Audit Report

1 Executive Summary

1.1 Project Information

Description	The TokenLabs contract is a staking protocol on the IOTA ecosystem. It allows users to stake tokens into the protocol, minting cert tokens proportionally as proof of stake. Users can then redeem their staked assets and corresponding rewards by using the cert tokens to unstake. TokenLabs enables users to select specific validators or opt for a default validator set, with all staked tokens being periodically allocated to the designated validators based on the staking cycle.
Type	DeFi
Auditors	jolyon, s3cunda
Timeline	Mon Jan 12 2026 - Wed Jan 14 2026
Languages	Move
Platform	IOTA
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Tokenlabs-LLC/LST
Commits	9973d19b248be3fe96f6116655214fe75d49fe14 a7a27c7c08f3aa179cceedb4ecef544c8904ebc4

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV	liquid_staking_v2/Move.toml	3eec117529a8054d08bf9e805a626cc0671b3348
NPO	liquid_staking_v2/sources/native_pool.move	9b41caa60ea8b35aa1d64a190e4cb1fcfc8d610b
MAT	liquid_staking_v2/sources/math.move	3e33786530bea0bf2e11214a2a5eced6078faebd
VSE	liquid_staking_v2/sources/validator_set.move	b4fa924b341725064cf6fe61f5668221ebcfc8ce
OWN	liquid_staking_v2/sources/ownership.move	28f5ca4a61c6a53856a734c0c4e3ded3462936fc
CER	liquid_staking_v2/sources/cert.move	85e70afef62620f63ffa833c39b715d0ba357e4b

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	6	0
Critical	0	0	0
Major	0	0	0
Medium	2	2	0
Minor	3	3	0
Informational	1	1	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Tokenlabs to identify any potential issues and vulnerabilities in the source code of the Tokenlabs smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 6 issues of varying severity, listed below.

ID	Title	Severity	Status
MAT-4	Minimum Share Minting with Asymmetric Rounding May Inflate Share Supply	Medium	Fixed
NPO-1	Missing Events for State Changes	Minor	Fixed
NPO-2	Missing Version Checks in Added Entry Functions	Minor	Fixed
NPO-3	Missing Zero-Value Check in `add_pending`	Informational	Fixed
NPO-7	Missing update function on `collectable_fee`	Medium	Fixed
VSE-5	Incorrect Upper Bound Check for Validator Updates	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Tokenlabs Smart Contract :

Admin

- `change_min_stake` - Update minimum amount of stake.
- `change_base_reward_fee` - Update `base_reward_fee` variable.
- `change_max_validator_stake_per_epoch` - Update maximum stake amount per epoch for each validator.
- `update_validators` - Update validator set and corresponding priorities.
- `update_rewards_threshold` - Update `rewards_threshold` variable.
- `update_rewards_revert` - Update `total_rewards` and `collected_rewards` value.
- `update_rewards` - Update `total_rewards` , `rewards_update_ts` and `collected_rewards` value.
- `add_pending` - Deposit `IOTA` into pending.
- `collect_fee_new` - Collect protocol fees.
- `set_pause` - Pause/unpause the protocol.
- `migrate` - Migrate the protocol.

User

- `stake` - Stake `IOTA` into protocol distribute to all validators and get `CERT` .
- `stake_to_validators` - Stake `IOTA` into protocol distribute to specific validators and get `CERT` .
- `unstake` - Burn `CERT` then get `IOTA` .

- `rebalance` - Move staked `IOTA` from bad validators into other validators.

4 Findings

MAT-4 Minimum Share Minting with Asymmetric Rounding May Inflate Share Supply

Severity: Medium

Status: Fixed

Code Location:

liquid_staking_v2/sources/math.move#40

Descriptions:

The `to_shares` function enforces minting at least one share whenever `amount > 0`, even when integer division would normally result in zero:

sources/math.move

```
/// Convert IOTA to tIOTA shares
public fun to_shares(ratio: u256, amount: u64): u64 {
    let mut shares = ((amount as u256) * ratio) / RATIO_MAX;
    assert!(shares <= (U64_MAX as u256), E_U64_OVERFLOW);

    // Ensure at least 1 share if amount > 0
    if (amount > 0 && shares == 0) {
        shares = 1;
    };

    (shares as u64)
}
```

This results in unsafe upward rounding when minting shares.

Suggestion:

It is recommended to reject deposits that are too small to mint at least one share, avoiding upward rounding at the math layer.

Resolution:

The team adopted our advice and fixed the issue by removing the forced minimum-share minting logic and instead rejecting deposits that are too small to mint at least one share, which can be found at commit 8aaca7a1b6f5a5fbf4b5e2c5786a2771cc25e755.

NPO-1 Missing Events for State Changes

Severity: Minor

Status: Fixed

Code Location:

liquid_staking_v2/sources/native_pool.move#1

Descriptions:

In the `native_pool` module, several functions modify critical protocol states or trigger significant asset movements but do not emit corresponding events.
e.g. `update_validators` `update_rewards_revert` `update_rewards` `rebalance`

Suggestion:

It is recommended to event logging after a critical operational status change.

Resolution:

The team adopted our advice and fixed the issue by adding event emissions for critical state changes, which can be found at commit f457cc87c5e8566122072ef2e6d31b952176a6f5.

NPO-2 Missing Version Checks in Added Entry Functions

Severity: Minor

Status: Fixed

Code Location:

liquid_staking_v2/sources/native_pool.move#253 430 666

Descriptions:

Several entry functions that mutate `NativePool` state do not enforce the module's version check mechanism. Specifically, the following functions can be executed without validating version:

`update_rewards_revert`

`add_pending`

`set_pause`

Suggestion:

It is recommended to add a version check at the beginning of all state-mutating entry and public functions.

Resolution:

The team adopted our advice and fixed the issue by add a version check at the beginning of all state-mutating entry and public functions, which can be found at commit f457cc87c5e8566122072ef2e6d31b952176a6f5.

NPO-3 Missing Zero-Value Check in `add_pending`

Severity: Informational

Status: Fixed

Code Location:

liquid_staking_v2/sources/native_pool.move#430

Descriptions:

The `add_pending` entry function does not validate that the input coin has a non-zero balance. As a result, zero-value Coin objects can be passed into the function and processed and emits a `PendingValueAddedEvent`.

sources/native_pool.move

```
  public entry fun add_pending(self: &mut NativePool, coin: Coin<IOTA>, _operator_cap:
&OperatorCap) {
    let prev_value = balance::value(&self.pending);

    let coin_balance = coin::into_balance(coin);
    balance::join(&mut self.pending, coin_balance);

    event::emit(PendingValueAddedEvent {
      prev_value,
      new_value: balance::value(&self.pending),
    });
  }
```

Suggestion:

It is recommended to validate that the input coin balance is greater than zero before merging it into the pending balance and emitting the corresponding event.

Resolution:

The team adopted our advice and fixed the issue by validate that the input coin balance, which can be found at commit f457cc87c5e8566122072ef2e6d31b952176a6f5.

NPO-7 Missing update function on `collectable_fee`

Severity: Medium

Status: Fixed

Code Location:

liquid_staking_v2/sources/native_pool.move

Descriptions:

The `collectable_fee` variable is used in `collect_fee_non_entry` and, according to the code logic, appears to be intended as the storage location for protocol fees. However, this variable is only initialized to 0 and has no other mechanisms to update its value. As a result, the `collect_fee_non_entry` function always withdraws the entire balance from `collected_rewards` as rewards.

sources/native_pool.move

```
fun collect_fee_non_entry(
        self: &mut NativePool,
        wrapper: &mut IotaSystemState,
        ctx: &mut TxContext
    ): Coin<IOTA> {
        let available = balance::value(&self.collectable_fee);//<= always 0
        let mut fee_coin = coin::from_balance(balance::split(&mut self.collectable_fee,
available), ctx);

        // If buffer doesn't have enough, withdraw the rest from validators
        if (available < self.collected_rewards) {//<= always true
            let needed = self.collected_rewards - available;//<= entire collected_rewards
            let validators = validator_set::get_validators(&self.validator_set);
            coin::join(&mut fee_coin, unstake_amount_from_validators(self, wrapper, needed,
validators, ctx));
        };

        // Update accounting
        self.collected_rewards = self.collected_rewards - coin::value(&fee_coin);
```

```
    fee_coin
}
```

Suggestion:

It is recommended to redesign the update logic for this ambiguously defined variable.

Resolution:

The team has adopted our advice and fixed the issue by removing the unused `collectable_fee` variable, which can be found at commit f457cc87c5e8566122072ef2e6d31b952176a6f5.

VSE-5 Incorrect Upper Bound Check for Validator Updates

Severity: Minor

Status: Fixed

Code Location:

liquid_staking_v2/sources/validator_set.move#99

Descriptions:

The `update_validators` function enforces an upper bound on the number of validators being updated:

sources/validator_set.move

```
    let len = vector::length(&validators);
    assert!(len < MAX_VLDRS_UPDATE, E_TOO_MANY_VLDRS);
```

This check rejects updates where `len == MAX_VLDRS_UPDATE`, even though the constant name implies that updating up to `MAX_VLDRS_UPDATE` validators should be allowed. This results in an off-by-one validation error that unnecessarily restricts valid inputs.

Suggestion:

It is recommended to allow the validator count to be less than or equal to the MAX_VLDRS_UPDATE.

Resolution:

The team adopted our advice and fixed the issue by allow the validator count to be less than or equal to the MAX_VLDRS_UPDATE, which can be found at commit 956ffb1c8a9677f2b5050ba10d627060fd787e5d.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.