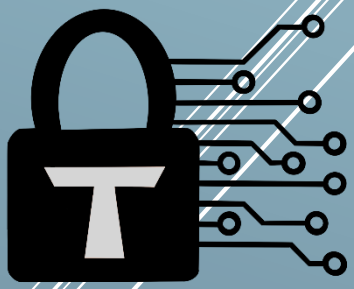
Trust Security



Smart Contract Audit

Story Protocol – Time Lock Vest Vault

13/02/2025

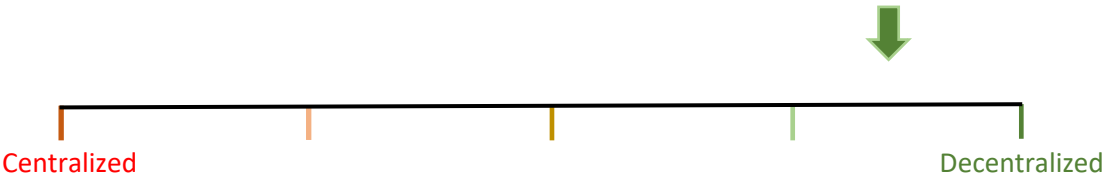
Executive summary

Category	Vault
Audited file count	3
Lines of Code	258
Auditor	rvierdiiev Lambda
Time period	05/02/2025 - 13/02/2025

Findings

Severity	Total	Fixed	Acknowledged
High	0	-	-
Medium	0	-	-
Low	3	3	0

Centralization score



Signature

Digitally signed by Trust
Date: 2025.02.13 20:37:06 Z00'00'

Document properties

Versioning

Version	Date	Description
0.1	06/02/2025	Audit of initial version
0.2	13/02/2025	Fix review and version 2

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- contracts/StakeRewardReceiver.sol
- contracts/TimelockVestVault.sol
- contracts/ValidatorWhitelist.sol

Repository details

- **Repository URL:** https://github.com/piplabs/timelock-vest-vault
- **Initial Commit hash:** 67543e2d0c982ba4c862ae36562ccf972b38c26c
- **Version 2 and Fix Commit hash:** 3bcc9ff3b4e2ee751f32c3ecf73ed5c2726a30c3

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

About the Auditors

Lambda is a security researcher and developer with multiple years of experience in IT security and traditional finance. This experience combined with his academic background in Mathematical Finance and Computer Science enables him to thoroughly examine even the most complicated code bases, resulting in several top placements in various audit contests.

rvierdiiev is a Web3 security researcher who participated in a large number of public audit contests on multiple platforms and has proven track record of experience.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project kept code as simple as possible, reducing attack risks
Documentation	**Good**	Overall the project is well documented.
Best practices	**Good**	Project uses battle-tested libraries and dependencies.
Centralization risks	**Excellent**	Project enables to enforce vesting schedules while keeping complete custody of the funds.

Findings

Low severity findings

TRST-L-1 Remainder not refunded

- **Category:** Logical flaws
- **Source:** contracts/TimelockVestVault.sol
- **Status:** Fixed

Description

The *_stake()* function in the staking contract (which is called by *stakeOnBehalf()*) returns a refund to the caller if the amount is not a gwei multiple:

```
        if (remainder > 0) {
            _refundRemainder(remainder);
        }
```

This is not handled and the (relatively small) amount will just be stuck in the vault. Moreover, the staked amount will be increased by the total amount, although not everything was staked.

Recommended mitigation

We recommend verifying that the user sends a gwei multiple and reverting otherwise.

Team response

Fixed in version 2 by letting the user withdraw any left tokens after the unlock duration.

Mitigation review

Fixed. The funds can now be retrieved after the unlock duration.

TRST-L-2 Off-by-one error when claiming unstaked tokens

- **Category:** Off-by-one errors
- **Source:** contracts/TimelockVestVault.sol
- **Status:** Fixed

Description

When claiming, it is checked if the **unstaked** amount is greater than the remaining one.

```
        if (unstaked >= amount) {
            stakeAgent.transferToVault(amount);
        } else if (availableInVault < amount) {
            uint256 remaining = amount - availableInVault;
            if (unstaked > remaining) {
                stakeAgent.transferToVault(remaining);
            } else {
```

```
                revert NotEnoughUnlockedTokens(remaining, unstaked);
            }
        }
```

However, when it is equal to the remaining one (meaning everything was requested), the transfer should still succeed.

Recommended mitigation

We recommend changing the comparison to **unstaked** >= **remaining**.

Team response

Fixed after the version 2 refactor.

Mitigation review

Fixed. In version 2, the claiming logic was changed and this issue no longer applies.

TRST-L-3 Unstake fee is paid by the contract

- **Category:** Input validation
- **Source:** contracts/TimelockVestVault.sol
- **Status:** Fixed

Description

The unstake fee is taken from the contract's balance:

```
        stakingContract.unstakeOnBehalf{ value: stakingContract.fee() }(
            _getStakeAgentAddress(beneficiary),
            validator,
            0,
            amount,
            ""
        );
```

While this may be intended (in which case the team must top up the contract with tokens for the allocations plus the fees), it can lead to problems: A user can unstake multiple times, in which case more fees can be deducted, leading to a situation where the remaining contract balance is not sufficient to cover all allocations.

Recommended mitigation

We recommend enforcing that the user sends enough tokens to cover the fees.

Team response

Fixed, the fee is now paid by the user.

Mitigation review

Fixed. The contract correctly enforces that the user pays the fee.

Additional recommendations

TRST-R-1 Unnecessary variable rewardClaimeds

The mapping **rewardClaimeds** is never used and can be removed.

TRST-R-2 Human-readable error for _getStakeableAmount()

The function *_getStakeableAmount()* will revert when the unlocked amount + the staked amount is greater than the allocation:

```
    function _getStakeableAmount(bytes32 beneficiary) internal view returns
(uint256) {
        // formula: allocation - unlocked - staked
        return
            allocations[beneficiary] -
            _getUnlockedAmount(beneficiary, uint64(block.timestamp)) -
            totalStakedAmounts[beneficiary];
    }
```

This for instance happens after a month when a user initially stakes the whole allocation. While reverting is fine in such situations, we recommend adding an explicit error (as in other code paths) for this scenario.

Systemic risks

TRST-SR-1 Dependency on the Story Blockchain

The contracts are auxiliary helper contracts for staking on the Story Blockchain. Any error or vulnerability in the staking logic will therefore have an impact on their correct functioning.