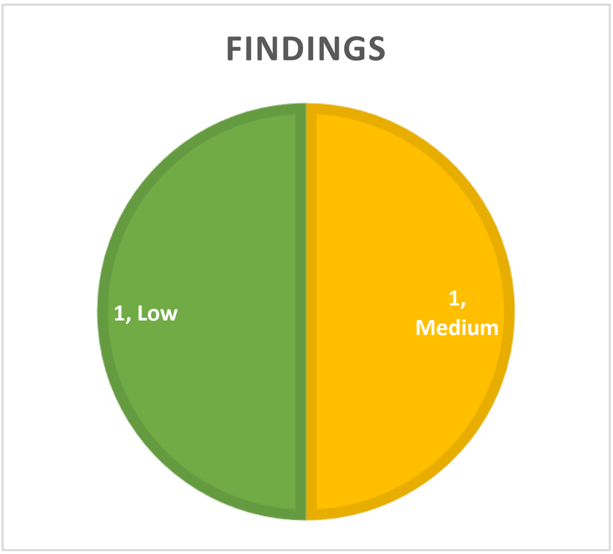
# Trust Security

Smart Contract Audit

Shapeshift rFOX

03/06/24

## Executive summary

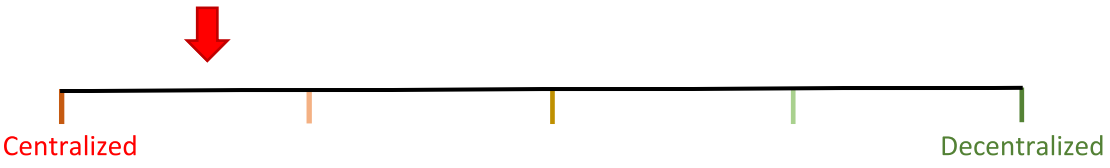


| | |
|---|---|
| Category | Staking |
| Audited file count | 3 |
| Lines of Code | 262 |
| Auditor | Trust |
| Time period | 23/05-29/05 |

Findings

| Severity | Total | Fixed | Acknowledged |
|---|---|---|---|
| High | - | - | - |
| Medium | 1 | 1 | - |
| Low | 1 | - | 1 |

Centralization score



Signature

Digitally signed by Trust
Date: 2024.06.03 14:58:18 +03'00'

# Document properties

## Versioning

| Version | Date | Description |
|---|---|---|
| 0.1 | 29/05/24 | Client report |
| 0.2 | 03/06/24 | Mitigation review |

## 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

- StakingV1.sol
- StakingInfo.sol
- UnstakingRequest.sol

## Repository details

- **Repository URL:** https://github.com/shapeshift/rFOX
- **Commit hash:** 1be689c87aaeba2b47af2a2b78ba92b5425d2177
- **Mitigation review hash:** bd94f6460452c16fc87225e3c60f5b3179cbda7c

## 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

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys assessing bounty contests in C4 as a Supreme Court judge.

## Disclaimer

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 | **Excellent** | Project kept code as simple as possible, reducing attack risks |
| Documentation | **Excellent** | Project is very well documented. |
| Best practices | **Good** | Project mostly adheres to industry standards. |
| Centralization risks | **Mediocre** | Project introduces significant trust assumptions for users |

# Findings

## Medium severity findings

### TRST-M-1 Insufficient padding can lead to precision loss and reduced rewards

- **Category:** Precision loss errors
- **Source:** StakingV1.sol
- **Status:** Fixed

#### Description

In rFOX, current rewards are calculated in *rewardPerToken()* function, which increases at a rate of **REWARD_RATE** (1e9) per second.

```
function rewardPerToken() public view returns (uint256) {
    if (totalStaked == 0) {
        return rewardPerTokenStored;
    }
    return
        rewardPerTokenStored +
        (((block.timestamp - lastUpdateTimestamp) * REWARD_RATE * WAD) /
            totalStaked);
}
```

Note that the larger the staked amount, the smaller **rewardPerTokenStored** increment becomes, to the point where if the denominator is larger than the numerator, rewards are lost entirely. Assuming per-second accrual, the condition for loss of rewards is:

totalStaked > 1 * 1e9 * 1e18 =>

totalStaked > 1e27

For 18 decimal tokens, this would be over 1e9 tokens being staked, which is not impossible. Also, the protocol aims to work with any compliant ERC20, which includes larger decimal tokens.

#### Recommended mitigation

The numerator should be padded with WAD squared. That would make precision loss extremely unlikely. Make sure to divided by WAD squared in the *earned()* calculation. A quick check shows the max uint256 (~1e77) can never be overflown.

#### Team response

“While we do anticipate potentially using these contracts for other tokens, we do not currently anticipate doing so for tokens with more than 18 decimals places. There was a similiar test for staking up to 900 million FOX tokens (~90% of the total supply) to ensure there was no precision loss. Either way, we think the fix makes the contracts more future proof.”

#### Mitigation review

The issue has been resolved by storing reward points in 1e27 format.

## Low severity findings

### TRST-L-1 Race condition during unstaking could cause user to lock for longer than intended

- **Category:** Race condition issues
- **Source:** StakingV1.sol
- **Status:** Acknowledged

**Description**

In rFOX, the owner may set a new unstaking cooldown period at any time. Suppose a user calls *unstake()*, when current cooldown is X, but due to misfortune owner increases the cooldown period before *unstake()* is executed. User would now be locked into a longer cooldown (without rewards) than they expected.

**Recommended mitigation**

There are two plausible ways to address this class of issues:

- Configuration changes could be made to require the protocol being paused. Then, a race condition won't occur as the user's action would revert.
- User's TX could pass a commitment to the current configuration set, which is verified when the TX is executed.

**Team response**

"The team acknowledges this issue but doesn't intend to resolve it. We believe the risk to be very low and any changes to the cooldown would have to be done through governance with plenty of warning. Pausing the contracts actually would not work because the transaction could still be waiting in the mempool with a lower gas price as the contract was paused, the cooldown increased, and unpaused. So the only full mitigation would be for the user to submit the expected cooldown period with their unstake call and otherwise revert. This could be done and may be in the future if we discover the frequency of these types of in-flight collisions to be problematic enough."

## Additional recommendations

### TRST-R-1 Emit events in global state-changing functions

In rFOX, there are several global parameters which can be configured by the owner. We recommend to emit an event to increase the visibility and transparency of the protocol.

### TRST-R-2 Separate rune address changes from staking

The *stake()* function accepts a rune address which will receive rewards. Note that it will override the previously set address:

```
info.stakingBalance += amount;
info.runeAddress = runeAddress;
```

This behavior could be surprising in some scenarios, for example if user assumes the earlier staking rewards will go to the previous rune address. It is recommended to separate out the address changing functionality to *setRuneAddress().* The *stake()* function could still keep the parameter for use when the current rune address is zero (first stake), so that user doesn't have to submit two transactions.

### TRST-R-3 Coding best practices

- The **initializer** function initializes all parent classes, except ReentrancyGuardUpgradeable. It is not strictly necessary since the default state of the guard is unlocked, but it's still considered a best practice.
- Internal functions (*updateReward*) should be preceded with _, to improve readability.
- Validate global state changes are not no-ops. For example, *unpauseUnstaking()* should verify **unstakingPaused** is true.

### TRST-R-4 Missing rune changing event in *stake()*

When a user changes their rune address through *setRuneAddress()*, it emits the SetRuneAddress() event. However, if the addresses changes in the *stake()* function, it is not emitted, which could create visibility issues.

### TRST-R-5 Follow Checks-Effects-Interactions in *stake()* function

The *stake()* function updates the user's rune address and staking balances after transferring the ERC20 token into the contract. It is recommended to follow CEI policies to reduce user callback risks. The risk level is very low here as the function is already guarded with a reentrancy guard, but it is still important in the context of read-only reentrancy safety with other contracts.

## Centralization risks

### TRST-CR-1 Owner has complete control of the contract funds

The contract owner can upgrade the implementation at any time. If the admin were to be compromised, any funds staked in the contract could be lost.

### TRST-CR-2 Owner can pause any functionality

The contract owner could use the *pause()* functions to disable any of the functionality in the contract, until they unpause it back.

### TRST-CR-3 Owner can set cooldown period arbitrarily

There is no upper bound to the cooldown period for unstaking. Further, a malicious admin could sandwich a user's unstaking transaction to make it higher than for other users.

### TRST-CR-4 Rewarding mechanism is off-chain

Rewards in rFOX are processed off-chain. A script snapshots user points at every epoch interval and translated them to rewards sent on THORchain. All details around time of snapshotting, conversion and dispatch of rewards, are trusted.