Audit Report

StarTerra Smart Contracts

v1.0

December 29, 2021

Table of Contents

License

Disclaimer

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 Terraform Labs to perform a security audit of a subset of the StarTerra 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 behaviour.
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/starterra/app-smart-contracts

Commit hash: `34ba139454860211f82e9dd94f2f38d975f6f27a`

The scope for the audit has been limited to the following modules:

```
contracts/staking
contracts/vesting
contracts/multisig
```

Commit hash: `2c3c8d550ced1511888faddd8ed0e8b73df64dab`

```
contracts/ido
```

`contracts/ido-prefund`

Commit hash: `9a50275da2492e3bba0ffc9d7be4388ea92db693`

including the files referenced by these modules in `packages/*`

NOTE: This audit covers the above contracts only and the overall codebase was still under active development at the time of the audit.

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 audited smart contracts implement the staking and reward functionalities for the StarTerra protocol, a vesting mechanism, a multisig smart contract, and an IDO solution.

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**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Withdrawal of IDO prefund contributions will fail in most cases due to funds being deposited in Anchor	Major	Acknowledged
2	Missing input validation on staking config update might lead to inconsistent state	Major	Resolved
3	The staking contract owner can withdraw all reward tokens from the contract but users can still not unbond without penalty	Minor	Acknowledged
4	Staking contract can run out of funds	Minor	Acknowledged
5	Multisig implementation does not follow CW3 specification	Minor	Resolved
6	Multisig proposals do not expire and cannot be closed	Minor	Resolved
7	Staking contract sends UST fees to arbitrary burn address	Minor	Acknowledged
8	Reward withdrawals in staking contract charge fees even if no rewards are distributableStaking	Minor	Resolved
9	Missing tax deductions on UST transfers	Minor	Resolved
10	IDO end date can be set in the past	Minor	Resolved
11	Missing bounds on fees in staking contract	Informational	Acknowledged
12	Multisig does not keep track of original proposer	Informational	Resolved
13	Unnecessary iteration over whole distribution schedule in reward calculation	Informational	Resolved
14	Owner privileges are checked twice	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	High	-
Level of Documentation	Medium	The project provides detailed user-facing documentation. However, the smart contract architecture is not documented in detail and the source code is sparsely commented.
Test Coverage	Medium-High	-

Detailed Findings

1. Withdrawal of IDO pre-fund contributions will fail in most cases due to funds being deposited in Anchor

Severity: Major

The function `withdraw_prefund` in `contracts/ido-prefund/src/execute.rs` will revert in most cases since most of the funds are not actually stored in the contract itself but are deposited into Anchor as soon as a certain threshold is reached. This means that withdrawal requests by end-users will fail until the funds are redeemed from Anchor into the contract.

Recommendation

Status: Acknowledged

Team reply: *"We had such a feature in our contract, but decided to remove it to protect our participants from anchor withdrawal fees. Currently, the balance of the contract is monitored and as soon as it drops below 100k ust, funds are automatically withdrawn from anchor."*

2. Missing input validation on staking config update might lead to inconsistent state

Severity: Major

The function `update_config` in `contracts/staking/src/contract.rs` fails to validate the unbonding configuration provided as an argument by not calling the available `assert_unbond_config` function, which ensures that unbonding periods are specified in the right order to be processable by the unbonding logic. This could lead to inconsistent contract state and errors during unbonding attempts.

Recommendation

Call `assert_unbond_config` in `update_config`.

Status: Resolved

3. The staking contract owner can withdraw all reward tokens from the contract but users can not unbond without penalty

Severity: Minor

The staking contract owner can use the function `emergency_withdraw` to remove all funds from the smart contract. This seems to be intended for emergency use in case of a vulnerability being detected. However, this does leave the user in a position in which they cannot use `unbond` to get their own funds back without a penalty.

Recommendation

Consider removing unbonding penalties when `emergency_withdraw` is used. An easy way to achieve this is by asserting that the contract is paused first.

Status: Acknowledged

Team reply: *"This is a security mechanism and our aim is to not use it, since smart contracts on Terra are easily migratable owner can achieve whatever he wants with it"*

4. Staking contract can run out of funds

Severity: Minor

The staking contract relies on being funded since rewards are transferred from the contract's balance to the user. In the case of the contract running out of funds, users are left without rewards.

Recommendation

Consider minting the reward tokens when required. Alternatively, allow users to unbond without penalty when funds run out.

Status: Acknowledged

Team reply: *"The amount of tokens for rewards is fixed and sent to each staking contract during the token generation event, staking contract follow the approach described in whitepaper in terms of token economy."*

5. Multisig implementation does not follow CW3 specification

Severity: Minor

The implementation of the multisig contract is relatively close to the CW3 specification (https://github.com/CosmWasm/cw-plus/blob/main/packages/cw3/README.md). However, it does not follow the naming convention for the interface, which might lead to confusion.

In addition, the code implements basic functionality that is already implemented by the CW3 reference implementation (https://github.com/CosmWasm/cw-plus/tree/main/contracts/cw3-flex-multisig).

Recommendation

Consider making the smart contract interface compatible with the CW3 specification to avoid confusion. In addition, the CW3 reference implementation could be used to provide additional functionality, such as closing unsuccessful proposals.

Status: Resolved

6. Multisig proposals do not expire and cannot be closed

Severity: Minor

Proposals submitted to the multisig remain in an executable state forever. This could lead to old proposals being executed by a signer after circumstances have changed, for example, when a transaction is not relevant anymore.
Additionally, the lack of support for cancelling proposals means that a proposer cannot withdraw a past proposal even if the set of signers changes through a config change.
An example of this being a problem could be a proposal having been replaced by a newer version and then accidentally being executed when the signing threshold is lowered.

Recommendation

Consider allowing proposers to close proposals and/or adding an expiry time for proposals, after which they cannot be executed anymore.

Status: Resolved

7. Staking contract sends UST fees to arbitrary burn address

Severity: Minor

The staking contract charges a UST fee on all operations. Collected fees can be withdrawn from the contract by sending them to a “burn address” that can be set by the owner to any address. This is inconsistent with the “burn” terminology.

Recommendation

We recommend clarifying the name of the burn address. If fees are to be burned, consider using a hardcoded real burn address. If fees are to be withdrawn by the owner, change the naming from burn address to `fee_recipient_address` to avoid confusion.

Status: Acknowledged

Team reply: *"This name follow our convention and statements in whitepaper."*

8. Reward withdrawals in staking contract charge fees even if no rewards are distributable

Severity: Minor

The `withdraw` function in `contracts/staking/src/contract.rs` does not fail when no rewards are distributable. This means that the user spends gas and fees without receiving any reward in return. Whilst this may be intentional, it is not what a user would expect as normal behaviour.

Recommendation

Consider returning a specific error when no rewards can be distributed, in order to abort the transaction and return the user's fees.

Status: Resolved

9. Missing tax deductions on UST transfers

Severity: Minor

In `contracts/ido-prefund/src/execute.rs` UST is sent out from the contract in the functions `withdraw_prefund` and `withdraw_ido_funds`. In both cases the code fails to account for Terra taxes on native tokens, meaning that these taxes are taken from the contract's balance.
In addition, functions `deposit_prefund` and `deposit_anchor_by_admin` send UST to Anchor. However, the bookkeeping of user funds takes into account taxes on UST transfers. This is unlikely to lead to the contract ever running out of funds due to fees. However, operator benefits may be slightly lower than expected in certain cases.

Recommendation

Consider deducting taxes in internal bookkeeping.

Status: Resolved

10. IDO end date can be set in the past

Severity: Minor

In `contracts/ido/src/execute.rs` functions `update_config` and `instantiate`, there are no checks to ensure that the end date is in the future. This could lead to accidentally configuring an IDO with an end date in the past.

Recommendation

Consider adding checks to ensure that the end date is greater than the current block time.

Status: Resolved

11. Missing bounds on fees in staking contract

Severity: Informational

The staking contract charges fees on almost all operations. These can be set arbitrarily by the contract owner through the `update_config` function. This could lead to the fees being set prohibitively high, rendering the contract impractical or impossible to use. If the contract is meant to be owned by governance, an accidental misconfiguration being voted in could take a long time to correct since it would require a new voting period.

Recommendation

Consider placing upper bounds on fees to protect users from prohibitive fees.

Status: Acknowledged

Team reply: *"Governance is not meant to own the staking contract, we are going to use multisig as contract admin."*

12. Multisig does not keep track of original proposer

Severity: Informational

The multisig contract does not keep track of the original proposer of proposals submitted. In `contracts/multisig/src/state.rs` the following data structure tracks proposals:

```
pub struct Proposal {
    pub messages: Vec<RawMessage>,
    pub signed_by: Vec<CanonicalAddr>,
    pub executed: bool,
}
```

At first, the original proposer is the first element in the `signed_by` vector, since this is where the proposer's signature is inserted. However, since signatures can be revoked, the first element might disappear. Keeping track of the original proposer is a useful feature for applications, but would also allow providing support for solving issue Multisig proposals do not expire and cannot be closed.

Recommendation

Consider adding a `proposer` field to the above data structure.

Status: Resolved

13. Unnecessary iteration over whole distribution schedule in reward calculation

Severity: Informational

In `contracts/staking/src/contract.rs` the function `compute_reward` iterates over the distribution schedule data structure when calculating rewards. However, since this vector is required to be ordered, the loop can be stopped once the block time has passed an end date since no remaining element will add to the reward being distributed.

Recommendation

Consider checking for the block time being greater than the end date and breaking out of the iteration.

Status: Resolved

14. Owner privileges are checked twice

Severity: Informational

In `contracts/ido/src/contract.rs` the function `assert_owner_privilege` is used to ensure that only the owner can execute the `UpdateConfig` message. However, this precondition is checked again in the `update_config` function in `contracts/ido/src/execute.rs`.

Recommendation

Consider removing one of the checks to reduce the number of storage reads.

Status: Resolved