Audit Report

Terra Liquidity Bootstrapping Pool

v1.0

December 23, 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 Delphi Labs Global Partners LLP to perform a security audit of the Terra Liquidity Bootstrapping Pool 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/astroport-fi/terraswap-lbp

Commit hash: `a306cc498f3b55683dfd589afd4e94b08fdc32bf`

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 code implements liquidity bootstrapping pool based on the Balancer model. This code review focuses on updates implementing the changes necessary to comply with the Terra network's Columbus-5 update.

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	Pools with big but different token amounts allow attackers to extract free value with minimal cost	Critical	Resolved
2	Duplicate storage in two contracts could lead to inconsistencies	Minor	Resolved
3	Migration of the pair contract is disabled	Minor	Resolved
4	A pair's asset infos are no longer stored sorted which might break other contracts	Minor	Acknowledged
5	Unnecessary sub-messages introduce complexity	Informational	Resolved
6	Contract name is not unique	Informational	Resolved
7	Pair contract registration message in factory can be replaced with sub-message reply	Informational	Resolved
8	Post initialize message in pair contract can be replaced with sub-message reply	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium-High	-
Code readability and clarity	Medium	-
Level of Documentation	Medium-High	-
Test Coverage	Medium-High	-

Detailed Findings

1. Pools with big but different token amounts allow attackers to extract free value with minimal cost

Severity: Critical

For big amounts of tokens in the offer and ask pools with different token values, the `calc_out_given_in` function in `contracts/terraswap_pair/src/math.rs:11` applies rounding, which opens a way for an attacker to extract value from a pool with a very small cost.

As an example, imagine a pool with `5_000_000_000_000` A tokens, and a pool with `1_000_000_000` B tokens, with both weights set to `1` for simplicity. If a user now sends `1` B token, we expect the user to get `5_000` A tokens back. When a user sends `1` A token, we expect the user to get `0` B tokens back (actually `0.0002`, but since we are dealing with integers here, the remainder will be dropped). The current implementation incorrectly returns `1` B token though. Imagine further that the value of `1` B token is `5_000` USD, and the value of `1` A token is `1` USD, an attacker can now get a risk free return of around `4_999` USD (minus transaction fees) per transaction. If the attacker repeats this attack, they will be able to drain the pool. Even worse, whenever the attacker shifts the balance enough, other arbitrageurs will be able to extract value by bringing the pool back to the `5_000` to `1` ratio, allowing the attacker to repeat the attack from where they started.

Here is a failing test case demonstrating the example:

```
#[test]
fn compute_swap_rounding() {
    let offer_pool = Uint128::from(5_000_000_000_000_u128);
    let offer_weight = FixedFloat::from_num(1);
    let ask_pool = Uint128::from();
    let ask_weight = FixedFloat::from_num(1);
    let offer_amount = Uint128::from(1_u128);

    let return_amount = Uint128::from(0_u128);
    let spread_amount = Uint128::from(0_u128);
    let commission_amount = Uint128::from(0_u128);

    assert_eq!(
        compute_swap(offer_pool, offer_weight, ask_pool, ask_weight, offer_amount),
        Ok((return_amount, spread_amount, commission_amount))
    );
}
```

Recommendation

We recommend adjusting the calculation in the `calc_out_given_in` function to remove the rounding issue described above. We also recommend adding test cases to the `calc_out_given_in` and `calc_in_given_out` as well as `compute_swap` and `compute_offer_amount` functions with a wide coverage of edge cases to ensure no other rounding issues exist. A simple way to achieve this is by using a fuzzing library.

Status: Resolved

2. Duplicate storage in two contracts could lead to inconsistencies

Severity: Minor

Both factory and pair contracts store the information about pairs, i. e. `asset_infos`, `contract_addr`, `liquidity_token`, `start_time` and `end_time`, in `contracts/terraswap_factory/src/state.rs:19` and in `contracts/terraswap_pair/src/state.rs:4`. This duplicate storage might lead to inconsistencies between the two contract states.

Recommendation

Consider using queries from one central place instead of state duplication in two contracts for increased consistency and better maintainability.

Status: Resolved

3. Migration of the pair contract is disabled

Severity: Minor

In the `Instantiate` message for the pair contract, the `admin` field is set to `None` in `contracts/terraswap_factory/src/contract.rs:149`. This implies that pair contracts cannot be migrated.

Recommendation

Depending on the intention here, we recommend setting the admin field to the contract owner to allow migrations.

Status: Resolved

4. A pair’s asset infos are no longer stored sorted which might break other contracts

Severity: Minor

In the last version of TerraSwap LBP, the `asset_infos` stored in the `FactoryPairInfo` struct of the factory contract in `contracts/terraswap_factory/src/contract.rs:139` were sorted. That is no longer the case. That change is not a problem for the audited contracts, since they use a `pair_key` helper function that generates a key based on the sorted `asset_infos`. This change, however, might break other contracts that depend on the previous design of stored sorted `asset_infos`.

Recommendation

We recommend storing the `asset_infos` sorted to minimize the probability of breaking other contracts.

Status: Acknowledged

The Astroport team states that storing a pair in the order provided at creation is intentional as it matches the implementation in the TerraSwap contracts.

5. Unnecessary sub-messages introduce complexity

Severity: Informational

Most contract interactions in the codebase are utilizing sub-messages, which have been introduced to Terra with the Columbus-5 upgrade. Sub-messages have been added to allow processing of the result of a call, for example, to handle errors. If the result of a call is not processed, regular messages should be used. There is no security concern in the usage of sub-messages, since they currently use the `ReplyOn::Never` value which causes a failure to propagate to the caller.

Unnecessary usage of sub-messages can be found in:

- `contracts/terraswap_factory/src/contract.rs:33`, `145` and `172`
- `contracts/terraswap_pair/src/contract.rs:73`, `101`, `259`, `307`, `360`, `370`, `381` and `492`
- `contracts/terraswap_router/src/contract.rs:117` and `141`
- `contracts/terraswap_router/src/operations.rs:45`, `59` and `97`

Recommendation

We recommend using regular messages in the cases above to stick to best practices.

Status: Resolved

6. Contract name is not unique

Severity: Informational

In `contracts/terraswap_token/src/contract.rs:16`, the contract name from the CW20 base contract is used in the following string declaration:

```
const CONTRACT_NAME: &str = "crates.io:cw20-base";
```

Recommendation

We recommend using a name unique to TerraSwap LBP.

Status: Resolved

7. Pair contract registration message in factory can be replaced with sub-message reply

Severity: Informational

In `contracts/terraswap_factory/src/contract.rs:156`, a hook pattern is used to register the newly created pair's contract address in the factory. That pattern requires an exposed `Register` message type in the factory contract. Sub-messages have been introduced with the Columbus 5 upgrade of Terra to allow the processing of a reply without the need to expose a public message handler.

Recommendation

We recommend replacing the hook pattern with a sub-message reply. At the same time, we do not recommend the removal of the hooks from the messages, since that could break other contracts.

Status: Resolved

8. Post initialize message in pair contract can be replaced with sub-message reply

Severity: Informational

In `contracts/terraswap_pair/src/contract.rs:86`, a hook pattern is used to set the newly created liquidity token contract address in the pair contract's config. That pattern requires an exposed `PostInitialize` message type in the pair contract.

Recommendation

As above, we recommend replacing the hook pattern with a sub-message reply. At the same time, we do not recommend the removal of the hooks from the messages, since that could break other contracts.

Status: Resolved