Audit Report

Altered Protocol

August 16, 2021

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by LoTerra to perform a security audit of Altered Protocol.

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 behavior.
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/LoTerra/altered-protocol

Branch: `0.10`

Commit hash: `3f07fa21149f1ae7877d5c30f673a27bbf33f7fb`

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

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	Minting/burning of tokens for/from a specific account will change all other account balances	Critical	Resolved
2	Zero liquidity of UST/Altered TerraSwap pair will lead to rebase failing, potentially causing a sell-off	Major	Resolved
3	Overflow checks not set for release profile	Minor	Resolved
4	CW2 version information uses wrong contract name	Minor	Resolved
5	Code repetition in rebase functions increase complexity and might make extension difficult	Informational	Acknowledged
6	Unnecessary human to canonical address conversions increase gas usage	Informational	Resolved

Code Quality Criteria

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

Detailed Findings

1. Minting/burning of tokens for/from a specific account will change all other account balances

Severity: Critical

In the current implementation, an account's balance is calculated by multiplying the total supply by the index stored for that account. Each account's index is stored as a ratio between the last balance as the numerator and the last total supply as the denominator, see for example `src/helpers.rs:108-109`. Any changes to the total supply do not update those stored indexes.

This implementation allows the balance for each account to fluctuate with the total supply, but it leads to issues when trying to mint/burn tokens for/from a specific account:

During the mint call handler, the total supply is increased by the minted amount in `src/contract.rs:250`. Since the indexes of accounts are not updated, every account will get a balance increase proportional to the change in total supply. The recipient of the minted tokens will also get the minted balance assigned in `src/contract.rs:259-263`, which means that the sum of all account balances is increased by two times the minted amount and hence bigger than the updated total supply value.

Likewise, during burns, the total supply is reduced in `src/contract.rs:210`, but again, the indexes of all accounts except the burner's account are not adjusted. Consequently, all accounts will proportionally have their balance reduced. Additionally, the burner's account balance will be reduced by adjusting the index for that burned amount in `src/contract.rs:203-207`. This implies that the sum of all account balances will decrease by two times the burned amount, which is more than the updated total supply value.

Recommendation

We recommend changing the storage architecture to only store the share for each account, and store the sum of all shares globally. The balance of each account can then be calculated by multiplying the total supply with the share of the account divided by the sum of all shares. This allows global adjustments of the total supply to increase/reduce all account balances as well as minting/burning by adjusting the shares of the recipient/burned account, the sum of all shares and the total supply.

Alternatively, minting and burning could be removed or disabled from the contract.

Status: Resolved

2. Zero liquidity of UST/Altered TerraSwap pair will lead to rebase failing, potentially causing a sell-off

Severity: Major

During the rebase, the current token ratio is calculated in `src/rebase.rs:13`. That calculation divides the current amount of UST tokens by the current amount of Altered tokens on TerraSwap. When the amount of Altered tokens is zero, for example if liquidity providers withdraw all of their liquidity because of a black swan event, rebasing will panic in `src/rebase.rs:55`. Without a rebase, the Altered token becomes inelastic and will destabilize. Worst case, this could lead to a sell-off of the Altered token.

Recommendation

We recommend adding a boolean flag to `TokenInfo` to disable any interactions with the Altered token if the liquidity of the UST/Altered TerraSwap pair ever reaches zero.

Status: Resolved

3. Overflow checks not set for release profile

Severity: Minor

`Cargo.toml` does not enable `overflow-checks` for the release profile. That implies that integer over-/underflows can occur, which will lead to inconsistent information. For example, the total balance when creating accounts in `src/contract.rs:80`, minting new tokens in line `250` or the allowance when increasing it in `src/allowances.rs:32` could overflow.

Recommendation

We recommend enabling overflow checks.

Status: Resolved

4. CW2 version information uses wrong contract name

Severity: Minor

The contract name for the CW2 version information in `src/contract.rs:27` states that the contract is called `crates.io:cw20-base`, which is the name of the CW20 base contract. That will cause issues, e. g. migration of the contract will fail in `src/contract.rs:424`.

Recommendation

We recommend setting an appropriate name for the contract, e. g. `crates.io:cw20-elastic`.

Status: Resolved

5. Code repetition in rebase functions increase complexity and might make extension difficult

Severity: Informational

The rebase functions `rebase_user_balance_and_supply_negative`, `rebase_user_balance_negative`, `rebase_user_balance_positive` and `rebase_user_balance_unchanged` in `src/helpers.rs` contain repetitive code. That increases the complexity of the code base and might make maintenance and extension of the contract more difficult.

Recommendation

We recommend removing repetitions in the codebase.

Status: Acknowledged

6. Unnecessary human to canonical address conversions increase gas usage

Severity: Informational

The human to canonical address conversions in `src/contract.rs:169` and `293` as well as `src/allowances.rs:135` and `180` are unnecessary, since the conversion already happened a few lines above and the result is stored in `sender_raw` and `owner_raw`, respectively. Those conversions are inefficient and increase gas usage.

Recommendation

We recommend using the available values in `sender_raw` and `owner_raw` instead.

Status: Resolved