Audit Report

Olympus Pro

v1.0

March 21, 2022

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 Olympus DAO to perform a security audit of Olympus Pro.

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/sandclock-org/terra-olympus-pro

Commit hash: `1b89f27d530ae479489dcac253b777d2294a0581`

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

Olympus Pro is a platform that helps protocols acquire their own liquidity. Olympus Pro provides "bonds-as-a-service", so protocols no longer need to pay out high incentives to rent liquidity. Instead of staking their LP (liquidity provider) tokens for farming rewards, users can exchange their LP tokens for a protocol's governance tokens at a discounted rate through a process called bonding.

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**.

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	Deposited funds remain locked in custom bond contract for perpetuity	Critical	Resolved
2	Fees sent to the incorrect treasury leads to loss of Olympus treasury income	Major	Resolved
3	Additional deposits will reset the vesting term of existing bonds	Minor	Acknowledged
4	Missing support for upgrades of bond and treasury contracts	Minor	Acknowledged
5	Centralization risk in withdrawal of treasury funds	Minor	Acknowledged
6	Usage of different versions of cw20 crate	Informational	Resolved
7	Inefficiency during permission check leads to extra storage read operation	Informational	Acknowledged
8	CW-Storage-Plus can simplify the code	Informational	Resolved
9	Setting minimum price to zero is unnecessary	Informational	Resolved
10	Duplicate bond price calculation in deposit function is inefficient	Informational	Resolved
11	Duplicate debt ratio calculation in deposit function is inefficient	Informational	Acknowledged
12	Canonical address transformations are inefficient	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium**	-
Code readability and clarity	**Medium**	-
Level of documentation	**Low-Medium**	The code has minimal documentation. The Olympus Pro documentation site provides an overview of the high-level functionality but does not provide specific details about the underlying mechanisms and implementation.
Test coverage	**Medium**	-

Detailed Findings

1. Deposited funds remain locked in custom bond contract for perpetuity

Severity: Critical

The custom_bond contract's deposit function at contracts/custom_bond/src/execute.rs:169 does not include any functionality to transfer the deposited principal token funds into the custom_treasury contract. Because of this, any deposited principal token funds will remain locked in the deposit contract for perpetuity.

Recommendation

We recommend adding a transfer message to the custom_bond contract in order to move funds to the custom_treasury contract.

Status: Resolved

2. Fees sent to the incorrect treasury leads to loss of Olympus treasury income

Severity: Major

During the bond creation using create_bond_from_temp and create_bond at contracts/factory/src/contract.rs:231 and 282, the factory contract is using the incorrect value of custom_treasury as olympus_treasury instead of config.treasury. The config.treasury treasury address provided during the initialization at contracts/factory/src/contract.rs:36 is currently unused.

The issue can be fixed through governance because the olympus_treasury address can be changed through the update_olympus_treasury function. But the collected fees would not be sent to the olympus_treasury until the address is correctly set, they would be transferred to the custom_treasury contract which Olympus does not own.

Recommendation

We recommend using config.treasury instead of custom_treasury as the value of olympus_treasury.

Status: Resolved

3. Additional deposits will reset the vesting term of existing bonds

Severity: Minor

Depositing additional funds to an existing bond will cause the vesting term of these existing deposits to be reset in `contracts/custom_bond/src/execute.rs:261`. That implies that funds that are not yet redeemed by the bonder will have an extended vesting period.

For example, if Alice deposits 100 token A (principal token) on January 1, the vesting term is 5 days and the bond price is 25, then she would get 4 payout tokens after the completion of 5 days, i.e on January 6. Suppose that after 4 days, i.e. on January 5 Alice deposits an additional 100 token A and Alice did not redeem during those 4 days, then her new vesting term will be again 5 days as per line `261`. If she wants to redeem after 1 day, i.e. on January 6, she will only be able to redeem `8 * 1 / 5 = 1.6` token B, even though the previous 4 tokens should have already been vested.

Recommendation

We recommend allowing vested tokens to be withdrawn by either allowing multiple `bond_info` per `bonder` or using a weighted calculation on redemption of tokens.

Status: Acknowledged

4. Missing support for upgrades of bond and treasury contracts

Severity: Minor

Creation of the bond and treasury contracts sets the factory contract as the admin in `contracts/factory/contract.rs:197`, `226`, and `277`. This means only the factory contract can upgrade the bond and treasury contracts, but there is no function in the factory contract to execute updates. The contracts also lack versioning information that can be handy in the future during upgrades of the contract.

Recommendation

We recommend implementing an explicit function in the `custom_factory` contract that supports an upgrade of the `custom_bond` and `custom_treasury` contracts. In addition, we recommend providing contract versioning information.

Status: Acknowledged

5. Centralization risk in withdrawal of treasury funds

Severity: Minor

Currently, access to the `withdraw` function in `contracts/custom_treasury/src/contract.rs:118` is controlled by the assert_policy_privilege function which ensures that the info.sender equals the config.policy address. The withdraw function allows the policy account to issue an arbitrary withdrawal from the treasury to any recipient. In the event of the policy account being compromised, an attacker would have the ability to completely drain the treasury.

We classify this finding as minor because while the impact would be major, the likelihood of this event occurring is low if strong key management practices are used on the policy account.

Recommendation

We recommend using a multisig for the policy account, together with following best practices on key management.

Status: Acknowledged

6. Usage of different versions of cw20 crate

Severity: Informational

Within the project's Cargo.toml files, different versions are used for the cw20 crate:

- Version 0.10.3 in contracts/custom_treasury/Cargo.toml:28
- Version 0.8.0 in contracts/custom_bond/Cargo.toml:29
- Version 0.8.0 in contracts/olympus_pro/Cargo.toml:20

Recommendation

We recommend using the most up-to-date version of the cw20 crate throughout the contracts. The newest version is 0.11.1.

Status: Resolved

7. Inefficiency during permission check leads to extra storage read operation

Severity: Informational

The assert_policy_privilege function in contracts/factory/src/contract.rs:50 reads the config value to enforce access control of functions, but calling functions read the config again during their execution. This results in the contract performing one extra storage read operation that could be avoided.

Recommendation

We recommend reading the config either outside or within the `assert_policy_privilege` function and passing it in or out of the function.

Status: Acknowledged

8. CW-Storage-Plus can simplify the code

Severity: Informational

Throughout the contracts, the older `cosmwasm_storage` storage functionality is used. The new storage functionality provided by `cosmwasm_storage_plus`, `Item` and `Map`, corresponds to the concepts of `Singleton` and `Bucket`, but requires less gas usage and provides helper functions that will simplify the code.

For example, in each of the contract's state files, the functions for reading/writing/removing singletons and buckets are explicitly defined. This functionality can be removed with the use of `Item` and `Map`.

Recommendation

We recommend replacing the usage of `cosmwasm_storage::singleton` and `cosmwasm_storage::bucket` with `cosmwasm_storage_plus::Item` and `cosmwasm_storage_plus::Map` respectively.

Status: Resolved

9. Setting minimum price to zero is unnecessary

Severity: Informational

In `contracts/custom_bond/src/execute.rs:306-307`, the custom bond contract's `state.terms.minimum_price` value is set to zero if the `bond_price` is not less than `state.terms.minimum_price`. There does not appear to be a reason why this value is set to zero.

Recommendation

We recommend removing the else condition in `contracts/custom_bond/src/execute.rs:306-307`.

Status: Resolved

10. Duplicate bond price calculation in deposit function is inefficient

Severity: Informational

The `deposit` function in `contracts/custom_bond/src/execute.rs:169` indirectly performs the `get_bond_price` calculation three times for every invocation.

The `deposit` function calls `get_true_bond_price` in `contracts/custom_bond/src/execute.rs:192` and `get_payout_for` in `213` and `222`. Each of these functions independently calls the `get_bond_price` function.

In addition, in lines `300-303`, `bond_price` is also calculated manually without calling the `get_bond_price` function, rather the code from the original function was duplicated. The following are specific references to the three occurrences where the duplication occurs:

- `get_true_bond_price` in `contracts/custom_bond/src/utils.rs:73`
- `get_payout_for` in `contracts/custom_bond/src/utils.rs:90`
- Duplicated code in `contracts/custom_bond/src/execute.rs:300-303`

Recommendation

We recommend calculating the bond price once for every invocation of the `deposit` function, and then reusing that value where necessary. This can be accomplished by directly calling the `get_bond_price` function and storing the price in a variable. The variable can then be passed into the `get_true_bond_price` and `get_payout_for` functions which can be refactored to remove the occurrences where they call `get_bond_price`. Additionally, the calculation in `contracts/custom_bond/src/execute.rs:300-303` can now be removed as it is redundant.

Status: Resolved

11. Duplicate debt ratio calculation in deposit function is inefficient

Severity: Informational

The `deposit` function in `contracts/custom_bond/src/execute.rs:169` indirectly performs the `get_debt_ratio` calculation five times for every invocation. The `deposit` function explicitly calls `get_debt_ratio` two times in lines `302` and `321`, and then `get_debt_ratio` is also called for every invocation of `get_bond_price` which is called three times.

Recommendation

We recommend calculating the debt ratio once for every invocation of the `deposit` function, and then reusing that value where necessary.

Status: Acknowledged

12. Canonical address transformations are inefficient

Severity: Informational

Usage of canonical addresses for storage is no longer recommended as a best practice. The reason is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit. Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend using the new `Addr` type instead of `CanonicalAddr`.

Status: Acknowledged