Audit Report

Zodiac Protocol Core Extension

v1.0

March 22, 2024

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.

THIS AUDIT REPORT IS ADDRESSED EXCLUSIVELY TO THE CLIENT. THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THE CLIENT OR THIRD PARTIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

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 Osmosis Grants Company to perform a security audit of Zodiac Protocol's Core 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 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 target:

Repository	https://github.com/zodiac-protocol/contracts
Commit	`4b1eeae1724fefc7e08447d2155fb79b76be899d`
Scope	In scope of this audit were all changes since our previous audit, which was performed at commit `ce8348ca265258bb0e51cef70087a32cafeaa818`.

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

Zodiac Protocol is a DeFi protocol that allows users to manage the risks associated with providing liquidity to AMMs (Automated Market Makers).
It deconstructs traditional LP tokens into Principal tokens, which remove the risk of volatile trading fees, and Yield tokens, which remove impermanent loss risk inherent to LPs.

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 improving 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 in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	High	The protocol uses Stargate messages to communicate with the underlying Cosmos SDK appchain.
Code readability and clarity	Low	-
Level of documentation	Medium	-
Test coverage	Medium-High	`cargo tarpaulin` reports a test coverage for the contracts in scope of 91.01%.

Summary of Findings

No	Description	Severity	Status
1	Missing principal token maximum supply validation	Minor	Resolved
2	Unimplemented state migration for `GENESIS_POOL_STATE` and `principal_token_max_supply`	Minor	Acknowledged
3	Incoherence in `TrackBeforeSend` and `BlockBeforeSend` response attributes	Informational	Resolved
4	Usage of deprecated `to_binary` function	Informational	Resolved
5	Usage of magic numbers decreases maintainability	Informational	Partially Resolved

Detailed Findings

1. Missing principal token maximum supply validation

Severity: Minor

The osmo_bal_lockup_vault contract does not perform validation before storing input data in the principal_token_max_supply configuration field. If a privileged user would by mistake or intentionally provide a very small or zero number, the contract would become unusable in practice.

The lack of validation exists during both instantiation in contracts/osmosis/zodiac_osmo_bal_vault/src/contract.rs:97 and configuration update in lines 647-649.

Recommendation

We recommend defining a minimum allowed value for principal_token_max_supply or at least enforcing it to be greater than zero.

Status: Resolved

2. Unimplemented state migration for GENESIS_POOL_STATE and principal_token_max_supply

Severity: Minor

In contracts/osmosis/zodiac_osmo_bal_vault/src/state.rs:17 and 41, the GENESIS_POOL_STATE and principal_token_max_supply states are introduced to support deposit limit and pool state snapshot.

However, the migration handler in contracts/osmosis/zodiac_osmo_bal_vault/src/contract.rs:50 does not perform any state migration to support them.

Consequently, migrating the old zodiac_osmo_bal_vault contract to the current version will cause the contract to fail to work correctly due to serialization and deserialization storage errors.

We classify this as a minor issue since it could impact existing deployments of previous versions of this contract.

Recommendation

We recommend implementing state migration for the `GENESIS_POOL_STATE` and `principal_token_max_supply` states.

Status: Acknowledged

3. Incoherence in `TrackBeforeSend` and `BlockBeforeSend` response attributes

Severity: Informational

In `contracts/osmosis/zodiac_osmo_bal_vault/src/contract.rs:195`, the `Response` generated for the `TrackBeforeSend` hook should be consistent with the one generated in the `BlockBeforeSend` hook.

However, the one returned in the `TrackBeforeSend` is missing the `("action", "no-op")` attribute.

Recommendation

We recommend enriching the `Response` of the `TrackBeforeSend` hook with the `("action", "no-op")` attributes.

Status: Resolved

4. Usage of deprecated `to_binary` function

Severity: Informational

In `contracts/osmosis/zodiac_osmo_factory/src/testing/integration.rs:14`, `packages/zodiac/src/utils.rs:170` and `176`, the `to_binary` function is used to serialize data structures to JSON bytes.

As the function is currently deprecated, its usage is not encouraged.

Recommendation

We recommend using the `to_json_binary` function in the aforementioned files.

Status: Resolved

5. Usage of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals are used. Using such “magic numbers” goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

A comprehensive list of hard-coded number literals has not been included in this report as many instances of `10000u32`, `2`, and `1` can be found in the contracts within scope.

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Partially Resolved