Audit Report

Yieldmos Outposts Osmosis

v1.0

May 17, 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 WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

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 Yieldmos Outposts Osmosis.

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/yieldmos/ac-outpost
Commit	`5fdef02bd583763898932494f5a3e6139ecd4cbe`
Scope	The scope was restricted to the `contracts/osmodca` and `contracts/osmostake` contracts.

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 Osmodca Outpost allows users to specify how they would like to manage their liquid OSMO balance. Users can select how much OSMO they would like applied to each execution and how frequently they would like it called (and how long they authorize the outpost to operate on their account with the `authz` module). It is intended to be called regularly by Yieldmos so that delegators can manage their rewards however they want.

The Osmostake Outpost allows users to specify how they would like to manage OSMO staking rewards by percentage. It is intended to be called regularly by Yieldmos so that delegators can manage their rewards based on their preference.

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	**Medium-High**	The protocol integrates with the Osmosis chain, the `authz` module, and the `authzpp` contract.
Code readability and clarity	**Medium**	Several instances of `TODO` comments, unfinished development, and compiler warnings are found in the codebase.
Level of documentation	**Medium-High**	Documentations are available at the `README` files and https://docs.yieldmos.com.
Test coverage	**Low**	The `instantiate_with_defaults` and `validator_only_compounding` integration tests fail.

Summary of Findings

No	Description	Severity	Status
1	Swap operations are vulnerable to sandwich attack	Critical	Resolved
2	Users can manipulate fee amounts in their favor	Critical	Acknowledged
3	Users can redirect tax fees to their address	Critical	Acknowledged
4	Incorrect denom validation in `Compound` message	Major	Resolved
5	Incorrect message ordering causes compounding to fail	Major	Resolved
6	Incorrect `token_in` parameter during simulations causes compounding to fail	Major	Resolved
7	Incorrect grant revocation in `osmostake` contract	Major	Resolved
8	Lack of validation during contract instantiation	Minor	Partially Resolved
9	Migration handlers allow downgrades and do not correctly store the contract version	Minor	Resolved
10	Dependencies affected by publicly known vulnerabilities	Informational	Acknowledged
11	Unused errors	Informational	Acknowledged
12	Multiple storage states are not exposed through smart queries	Informational	Acknowledged
13	TODO comments across the codebase	Informational	Acknowledged

Detailed Findings

1. Swap operations are vulnerable to sandwich attack

Severity: Critical

In `packages/osmosis-helpers/src/osmosis_swap.rs:384`, the Osmosis `MsgSwapExactAmountIn` message constructs the minimum amount of swap output (indicated as `token_out_min_amount`) value as zero. This is problematic as this essentially disables slippage protection.

Consequently, an attacker can perform a sandwich attack by purchasing the asset at the current price, forcing the `Compound` message to buy the asset at an increased price, and then immediately selling it for a profit. This attack can be performed repeatedly to cause a loss of funds for the user because the asset will be bought at an abnormally high price.

This issue is present across a few instances in the codebase:

- `packages/osmosis-helpers/src/osmosis_lp.rs:111`
- `packages/osmosis-helpers/src/osmosis_swap.rs:384`
- `packages/osmosis-helpers/src/osmosis_swap.rs:424`

Recommendation

We recommend modifying the `Compound` message to accept a minimum output amount parameter, similar to `contracts/osmodca/src/execute.rs:357-358`.

Status: Resolved

2. Users can manipulate fee amounts in their favor

Severity: Critical

In `contracts/osmostake/src/contract.rs:179`, the `Compound` message allows the caller to specify the `tax_fee` amount for the compound operation execution, which is the fee amount that the protocol will charge through the `split_rewards` function in the `authzpp` contract.

The issue occurs when users set the `max_fee_percentage` to a zero value amount to force `AuthorizedCompounder` to compound for free. Since the fee percentage is determined by the minimum value of the `max_fee_percentage` defined by the user and the granter-provided `percentage` value, the `min` function will evaluate zero as the base minimum, causing the protocol to receive zero tax fees.

We classify this issue as critical since the protocol loses fee income.

Recommendation

We recommend returning an error and reverting the transaction if the supplied `percentage` exceeds `max_percentage`.

Status: Acknowledged

The client states this is not an issue because users can only execute their own strategy. Users could then not pay any tax, which is fine because the compounder is not paying the gas fees. Since the compounder is the only entity authorized to run strategies on behalf of other users, the client can set both parameters.

3. Users can redirect tax fees to their address

Severity: Critical

In `contracts/osmostake/src/contract.rs:176-184`, the `Compound` message allows an `AuthorizedCompounder` to compound on behalf of users and charge the percentage of fee rewards depending on the `tax_fee` parameter. The `authzpp` contract will compute the rewards and distribute the `tax_fee` parameter's portion to the `taxation_address`.

However, since users can set `taxation_address` to any arbitrary address when creating the `Grant`, the user can set the address to themselves to receive the reward splits, causing the protocol to receive zero fees after compounding for the user.

We classify this issue as critical since the protocol loses fee income.

Recommendation

We recommend modifying the implementation so that `AuthorizedCompounder` specifies the `taxation_address` as a parameter.

Status: Acknowledged

The client states this is not an issue because users can only execute their own strategy. Users could then not pay any tax, which is fine because the compounder is not paying the gas fees. Since the compounder is the only entity authorized to run strategies on behalf of other users, the client can set both parameters.

4. Incorrect denom validation in `Compound` message

Severity: Major

In `contracts/osmodca/src/contract.rs:184`, the `Compound` message validates the compound token denom as `ujuno`. This is incorrect because the native token denom to auto-compound in the Osmosis chain is `uosmo`, not `ujuno`.

Consequently, the `Compound` message will always fail as users cannot send `ujuno` native tokens in the Osmosis chain.

Recommendation

We recommend modifying the denom to be `uosmo`.

Status: Resolved

5. Incorrect message ordering causes compounding to fail

Severity: Major

In `contracts/osmodca/src/execute.rs:221`, the `prefs_to_msgs` function appends the swap messages (indicated as `swap_msgs`) after the token transfer message (indicated as `send_msgs`). This is problematic because the transfer message intends to send the post-swapped token to a recipient, which will fail due to insufficient balance as the swap is not performed first.

This issue is present across several instances in the codebase:

- `contracts/osmodca/src/execute.rs:221`
- `contracts/osmodca/src/execute.rs:259`
- `contracts/osmostake/src/execute.rs:232`
- `contracts/osmostake/src/execute.rs:270`

Recommendation

We recommend using the `prepend_msgs` function to ensure swap messages are dispatched before transferring post-swapped tokens.

Status: Resolved

6. Incorrect `token_in` parameter during simulations causes compounding to fail

Severity: Major

In `packages/osmosis-helpers/src/osmosis_swap.rs:360`, the `generate_swap_and_sim_msg` function passes the `from_asset.denom` value to the `token_in` parameter when simulating the swap in line `303`. This is incorrect because `token_in` requires the value to include the token amount and token denom (e.g., `123uosmo` instead of `uosmo`) to query the `EstimateSwapExactAmountInRequest` message correctly.

Consequently, the auto-compounding will fail due to the simulation failure.

This issue is present across a few instances in the codebase:

- `packages/osmosis-helpers/src/osmosis_swap.rs:360`
- `packages/osmosis-helpers/src/osmosis_swap.rs:432`

Recommendation

We recommend passing the `token_in` parameter to include the token amount and token denom.

Status: Resolved

7. Incorrect grant revocation in `osmostake` contract

Severity: Major

In `contracts/osmostake/src/queries.rs:82-91`, the `query_revokes` function revokes the taxation grant as `AuthorizationType::SendAuthorization`. This is incorrect because the taxation grants in the `osmostake` contract are `AuthorizationType::GenericAuthorization` and `GrantRequirement::ContractExec`, causing the grant revocation to fail.

Recommendation

We recommend revoking the grant according to `contracts/osmostake/src/queries.rs:56-68`.

Status: Resolved

8. Lack of validation during contract instantiation

Severity: Minor

The `instantiate` functions, defined in `contracts/osmostake/src/contract.rs:31-77` and `contracts/osmodca/src/contract.rs:31-77`, respectively, for the `osmostake` and the `osmodca` contracts, perform validation of the `InstantiateMsg` message and then store its data in the storage.

However, the provided data is only partially validated. The `validate_addr` method does not check that the `staking_denom`, `destination_projects.denoms` and `destination_projects.swap_routes` do not contain empty strings.

Additionally, the functions do not validate whether the `destination_projects.denoms` contain duplicate values, which can cause state overwrites in the `KNOWN_DENOMS` storage item.

Similarly, deduplication is not applied to `destination_projects.swap_routes.osmo_pools` and `destination_projects.swap_routes.usdc_pools`, which can cause state overwrites in the `KNOWN_OSMO_POOLS` and `KNOWN_USDC_POOLS` storage items.

We classify this issue as minor because only the contract instantiator can cause it, which is a privileged address.

Recommendation

We recommend validating the `staking_denom`, `destination_projects.denoms` and `destination_projects.swap_routes` do not contain empty strings and deduplicating `destination_projects.denoms`, `destination_projects.swap_routes.osmo_pools` and `destination_projects.swap_routes.usdc_pools`.

Status: Partially Resolved

9. Migration handlers allow downgrades and do not correctly store the contract version

Severity: Minor

The `migrate` functions, defined in `contracts/osmostake/src/contract.rs:79-127` and `contracts/osmodca/src/contract.rs:79-127`, respectively, for the `osmostake` and the `osmodca` contracts, allow the contract migration admin to migrate the contract into a specific code ID and optionally update the project addresses via `MigrateMsg`.

However, the guard in line `84` does not error when the contract version to migrate (indicated as `storage_version`) is lower than `CONTRACT_VERSION`. The function allows the migration to downgrade the contract to a previous version and does not subsequently update the CW2 contract versions.

Consequently, the contract will have a mismatch between the actual and stored version and potential storage issues due to unintended downgrade.

We classify this issue as minor because only the contract migration admin can cause it, which is a privileged address.

Recommendation

We recommend returning an error when downgrading the contract version.

Status: Resolved

10. Dependencies affected by publicly known vulnerabilities

Severity: Informational

The `tungstenite` and `webpki` dependencies are used across packages in the contracts. As reported in CVE-2023-43669 and RUSTSEC-2023-0052, these dependencies are affected by a high-risk vulnerability with a 7.5 CVSS score.

Recommendation

We recommend updating the dependencies to their latest versions.

Status: Acknowledged

11. Unused errors

Severity: Informational

In several instances across the codebase, errors are defined but not used. This reduces the code readability and maintainability of the codebase. The following instances of unused errors have been detected:

- `contracts/osmodca/src/error.rs`
  - `OutpostError`
  - `OsmosisDestinationError`
  - `SailDestinationError`
  - `UniversalDestinationError`
  - `OsmosisHelperError`
  - `SemVer`
  - `InvalidCompoundPrefs`

    - `CheckedMultiplyFractionError`
    - `NotImplemented`
    - `SwapSimulationError`
    - `EncodeError`

- `contracts/osmostake/src/error.rs`
    - `OutpostError`
    - `AuthzppWithdrawTax`
    - `OsmosisDestinationError`
    - `SailDestinationError`
    - `UniversalDestinationError`
    - `OsmosisHelperError`
    - `SemVer`
    - `InvalidDCACompoundPrefs`
    - `InvalidCompoundPrefs`
    - `CheckedMultiplyFractionError`
    - `NotImplemented`
    - `SwapSimulationError`
    - `EncodeError`

Recommendation

We recommend using the above-mentioned `Errors` in the contract or removing them.

Status: Acknowledged

12. Multiple storage states are not exposed through smart queries

Severity: Informational

In `contracts/osmodca/src/contract.rs:195` and `contracts/osmostake/src/contract.rs:190`, the `query` entry points do not expose sufficient storage state values through smart queries. Specifically, the `PROJECT_ADDRS`, `KNOWN_OSMO_POOLS`, `KNOWN_USDC_POOLS`, and `KNOWN_DENOMS` storage states cannot be retrieved through smart queries.

Consequently, third-party contracts and nodes must perform a raw query to read the stored value, which is error-prone and decreases user experience.

Recommendation

We recommend implementing smart queries that expose the above-mentioned storage state.

Status: Acknowledged

13. TODO comments across the codebase

Severity: Informational

In several instances of the codebase, TODO comments are found:

- `contracts/osmodca/src/contract.rs:25`
- `contracts/osmostake/src/contract.rs:25`
- `packages/osmosis-destination/src/pools.rs:29`
- `packages/osmosis-destination/src/osmosis_lp.rs:278`
- `packages/osmosis-destination/src/osmosis_swap.rs:20,26`
- `packages/utils/src/msg_gen.rs:181`

This suggests that some improvements or functionality have not been implemented in these places. While this does not pose a security risk in itself, it may assist a potential attacker in creating attack vectors against the protocol.

Recommendation

We recommend resolving the TODO comments or removing them.

Status: Acknowledged