Audit Report

Osmosis Transmuter

v1.0

October 9, 2023

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 Osmosis Grants Company to perform a security audit of the Osmosis Transmuter CosmWasm smart contract.

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/osmosis-labs/transmuter
Commit	`58d156af55867088e294378b744394aadad0eab5`
Scope	All contracts were in scope.
Fixes verified at commit	`a4cfd53c105bfbd7e52d63b93337932431023a0b`

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 `transmuter` contract is designed to interact with the `cosmwasmpool` module, allowing for 1:1 swapping between multiple tokens with no fees.

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**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**High**	The provided documentation was comprehensive. Additional details on the type of expected assets and the legacy parameters required for compatibility would be desirable.
Test coverage	**High**	Tarpaulin reported test coverage of 98.27%

Summary of Findings

No	Description	Severity	Status
1	Lack of denom validation	Minor	Resolved
2	Missing validation steps when registering limiters	Minor	Partially Resolved
3	Unbounded loops could render main features unusable	Minor	Resolved
4	Zero-value swaps are allowed and introduce inefficiencies	Informational	Resolved
5	Unoptimized zero amount burning or minting	Informational	Resolved
6	Deregistering of non-existing limiters silently fails	Informational	Resolved
7	Missing Division invariant check	Informational	Resolved
8	Admin transfer procedure can be enhanced	Informational	Resolved
9	Redundant query function	Informational	Resolved
10	Lack of limiters could result in uncontrolled pool imbalance	Informational	Partially Resolved
11	Lack of role-based access controls for the pausing mechanism	Informational	Resolved
12	Default value conceals unexpected state	Informational	Resolved
13	Incorrect error message	Informational	Resolved
14	Typographical error in attribute naming	Informational	Resolved
15	Unused error messages	Informational	Resolved
16	Usage of panics for error handling	Informational	Resolved
17	Usage of vulnerable dependencies	Informational	Acknowledged

Detailed Findings

1. Lack of denom validation

Severity: Minor

Upon instantiation, the `transmuter` contract creates a pool with multiple denoms through the `TransmuterPool::new` function. However, the supplied denoms are not effectively validated in `contracts/transmuter/src/transmuter_pool/mod.rs:16-24`.

Incorrect denominations or an input typo would render some of the contract's features unusable, requiring a new instantiation and for the users to withdraw current deposits.

Recommendation

We recommend querying each denomination before storing it to ensure that the supplied string is a valid denomination, for example, by checking its metadata.

Status: Resolved

2. Missing validation steps when registering limiters

Severity: Minor

The `transmuter` contract lacks validation on some of the functions related to limiters.

Neither `upper_limit` nor `boundary_offset` undergo any validation when newly created in `contracts/transmuter/src/limiter/limiters.rs:63-64` and `207-209` or when updated in lines `314` and `348`. If a value of zero is provided, most trades will fail to satisfy the limiter's restrictions, making some features unusable. On the other hand, if a large number is set, the contract could become rapidly imbalanced as described in the issue Lack of limiters could result in uncontrolled pool imbalance

The `label` field is allowed to be empty upon registration of a new limiter in `contracts/transmuter/src/limiter/limiters.rs:287`. Although not a security risk, it can be misleading and affect the user experience.

Finally, when registering a new limiter the target denomination is not checked to be part of the current pool in `contracts/transmuter/src/limiter/limiters.rs:286-288`. This will render the limiter ineffective in case of a mistake or a typo, leaving the intended denomination without any control against imbalance.

Recommendation

We recommend validating both `upper_limit` and `boundary_offset` to be greater than zero and below a reasonable amount that effectively limits arbitrage.

In addition, we recommend validating that the trimmed `label` field is not an empty string. We also recommend validating that the denom of each new limiter is part of the pool assets.

Status: Partially Resolved

The client stated that they lacked data to decide on a reasonable upper limit and prefer to experiment without placing constraints that could make the pool unusable. The rest of the recommendations have been implemented.

3. Unbounded loops could render main features unusable

Severity: Minor

The `transmuter` contract includes a nested `for` loop structure in `contracts/transmuter/src/limiter/limiters.rs:391-394`. The outer loop iterates through all denominations specified during pool initialization while the inner loop iterates through all limiters registered by the admin. The size of both vectors can be arbitrarily decided by the admin.

If any of these vectors is excessively large, it can significantly inflate the gas cost for swap operations, pool joins, or exits, potentially causing operations to revert. The risk escalates when both arrays are overly extended, due to the quadratic computational complexity.

Although the comments in lines `365` and `378` state that the number of limiters is expected to be small, no actual mechanism is found to prevent the described scenario.

Recommendation

We recommend limiting the maximum number of limiters per denomination and denominations per pool.

Status: Resolved

4. Zero-value swaps are allowed and introduce inefficiencies

Severity: Informational

`contracts/transmuter/src/sudo.rs` defines the `SwapExactAmountIn` and `SwapExactAmountOut` messages which are used by the `Pool Manager` module of Osmosis to automatically execute swaps, including multi-hop swaps.

These messages contain the `token_in` and `token_out` `Coin` fields. If any of the `Coin` fields have an `amount` field set to zero, unnecessary swaps are performed. Unnecessary steps in a multi-hop swap could cause excessive gas costs for a user who might not be aware of the `transmuter` pool being used.

For example, the `SudoMsg::SwapExactAmountIn` message triggers such a redundant swap when `token_in` is set to `Coin::new(0, "ABC".to_string())` and `token_out_min_amount` is set to `Uint128::from(0)`.

Similarly, the `SudoMsg::SwapExactAmountOut` message will cause redundant swaps for any valid `token_in_max_amount` if `token_out` is set to a value such as `Coin::new(0, "XYZ".to_string())`.

These messages can be sent from the `Pool Manager` module automatically as part of a bigger swap route. For example, redundant parameter combinations from the out-of-scope `cosmwasmpool` implementation could cause this issue.

Recommendation

We recommend validating that the input and output amounts are not zero before executing a swap.

Status: Resolved

5. Unoptimized zero amount burning or minting

Severity: Informational

The functions `swap_tokens_for_alloyed_asset` and `swap_alloyed_asset_for_tokens`, defined in `contracts/transmuter/src/contract.rs:299` and `370`, allow minting or burning of zero alloyed tokens. These scenarios arise from `join_pool` and `exit_pool` entry points when a user passes a `Coin` structure with an `amount` set to zero.

Although the resulting submessage to `x/bank` module will fail, unnecessary computation and hence gas will be spent as both functions call `check_limits_and_update` which can be computationally intensive due to nested iterations inside.

Recommendation

We recommend validating that the amounts to mint or burn are not zero as early as possible to increase efficiency.

Status: Resolved

6. Deregistering of non-existing limiters silently fails

Severity: Informational

In `contracts/transmuter/src/contract.rs:187`, during limiter deregistration, the `deregister` function declared in `contracts/transmuter/src/limiter/limiters.rs:290` is called. It performs removal operations over the vector without any prior validation, taking `denom` and `label` as parameters. The returned value of this operation is the same when an element is removed from the vector and when there is no match.

As a consequence, users will receive identical output for existing and non-existing `(denom, label)` pairs. This could lead to situations where limiters intended to be removed but containing a typo may still be in place after the admin tries to remove them.

Recommendation

We recommend checking that the provided `(denom, label)` pair is part of the limiters in storage, raising an error if that is not the case.

Status: Resolved

7. Missing `Division` invariant check

Severity: Informational

In `contracts/transmuter/src/division.rs`, the `Division` structure is declared to carry two timestamps: `started_at` denoting the beginning of the span, and `updated_at` denoting the last modification time of the division's value.

Line `49` enforces that `updated_at` is not before `started_at`. However, it does not enforce that `updated_at` is not after the division's end which is defined as `started_at + division_size`. This missing invariant check makes reasoning about divisions more difficult.

Recommendation

We recommend enforcing `updated_at <= started_at + division_size` since only divisions adjacent to new blocks can be updated, not previous ones.

Status: Resolved

8. Admin transfer procedure can be enhanced

Severity: Informational

`contracts/transmuter/src/admin.rs` implements a two-step admin transfer pattern following best practices, requiring the receiving side to accept ownership via the `claim` function. However, there is no dedicated API to cancel a proposed transfer.

At present, the current contract admin can only cancel an unclaimed admin transfer by directing the privileges back to their own address, followed by claiming this mock transfer, which is inefficient.

Recommendation

We recommend creating a dedicated entry point to allow the cancellation of an admin privilege transfer and rejecting transfers to the current owner address.

Status: Resolved

9. Redundant query function

Severity: Informational

The transmuter contract includes two queriable functions that are effectively identical: `get_share_denom` in `contracts/transmuter/src/contract.rs:499` and `get_alloyed_denom` in line `510`.

Although not a security issue, having redundant code is unnecessary and can decrease maintainability.

Recommendation

We recommend removing one of these redundant functions.

Status: Resolved

10. Lack of limiters could result in uncontrolled pool imbalance

Severity: Informational

The `transmuter` contract enforces a 1-1 relation between its pool assets. The limiter feature is in place to avoid swift liquidity imbalances happening when specific market conditions open arbitrage opportunities. However, there is no minimum amount of limiters required.

Upon instantiation, no limiter is created, which could be intended to facilitate a fast initial liquidity provision. From that on, the lack of a limiter, or a malicious admin removing all the limiters through the `deregister_limiter` function in `contracts/transmuter/src/contract.rs:120-190`, open up unlimited arbitrage opportunities that may imbalance the pools to an undesirable level.

Recommendation

We recommend that after a reasonable amount of liquidity has been reached, corresponding to the initial fast liquidity provision, no more liquidity is accepted until a minimum of one limiter per denom is set.

In addition, a minimum of one limiter per asset in the pool should be enforced when the admin removes limiters.

Status: Partially Resolved

The client stated that they lacked data to set a fixed liquidity cap to enforce the first batch of limiters. They choose to experiment with flexibility on when to set the initial limiters. Our recommendation to enforce a minimum of one limiter when modifying limiters has been implemented.

11. Lack of role-based access controls for the pausing mechanism

Severity: Informational

The codebase implements a pausing mechanism, which is in line with best practices. However, all of the administrative functions of the contract are centralized in the `admin` role, which goes against the principle of least privilege.

Segregating the pauser role has the additional benefit of swifter reactions in case of need when assigned to an EOA compared to the admin that might be managed by a multisig or a governance contract.

Recommendation

We recommend implementing a separate pauser role that can turn on and off the pausing mechanism.

Status: Resolved

12. Default value conceals unexpected state

Severity: Informational

In `contracts/transmuter/src/limiter/division.rs:310-316`, the `compressed_moving_average` function manages cases where result computation is infeasible. This arises when the time span in focus has a length of zero, occurring when block time matches the start time of the earliest division remaining post-pruning.

However, such a division must not exist since the average is computed prior to the division set update. The function currently defaults to returning zero, masking a potential issue with division updates.

While this is not a security concern, maintaining an accurate state aids in the early identification and resolution of any unexpected states.

Recommendation

We recommend throwing an error instead of returning the default value if an unexpected state is reached.

Status: Resolved

13. Incorrect error message

Severity: Informational

The `transmuter` contract's `exit_pool` function checks if the provided `token_out` coins are part of the pool. If one of the assets is not part of the pool, it raises an incorrect `InsufficientPoolAsset` error instead of `InvalidPoolAssetDenom`.

Although not a security risk, incorrect or non-descriptive errors may mislead users.

Recommendation

We recommend returning the appropriate error.

Status: Resolved

14. Typographical error in attribute naming

Severity: Informational

The response for the `transfer_admin` entry point response carries attributes `"method"` and `"andidate"`, defined in `contracts/transmuter/src/contract.rs:777-779`. Presumably, `"candidate"` was the intended name.

Within CosmWasm, attributes play important roles in event logging, auditability, and interoperability. A typographical mistake might lead off-chain components or other modules to miss events.

Recommendation

We recommend correcting typographical errors to improve readability.

Status: Resolved

15. Unused error messages

Severity: Informational

In `contracts/transmuter/src/error.rs`, several error messages are defined but remain unused, specifically:

- `InvalidPoolAssetDenom` (see Incorrect error message)
- `FundsMismatchTokenIn`

Recommendation

We recommend removing unused error messages to improve the maintainability of the codebase.

Status: Resolved

16. Usage of panics for error handling

Severity: Informational

It has been noticed that the `expect` macro is used in `contracts/transmuted/src/limiter/limiters.rs:154` for the error handling mechanism.

The usage of `expect` is generally discouraged because it leads to panics without a developer-friendly error message. `expect` also causes the wasm execution to abort, which does not allow handling of the error from calling functions.

Recommendation

We recommend unifying the error handling mechanism in order to standardize individual parts of the code, which will increase readability and simplify the code development process in the future.

Status: Resolved

17. Usage of vulnerable dependencies

Severity: Informational

It was found that the codebase uses dependencies utilizing packages with known vulnerabilities. As reported in https://rustsec.org/advisories/RUSTSEC-2022-0093 and https://rustsec.org/advisories/RUSTSEC-2023-0052, the `ed25519-dalek` and `webpki` crates are affected by issues of high impact.

These vulnerabilities are not directly exploitable in a CosmWasm smart contract and do not affect any of the current code fragments. Therefore this issue has been raised with informational severity for completeness.

Recommendation

We suggest verifying that the current code development process does not include any vulnerable dependencies, as well as periodically checking publicly known issues in the dependencies used.

Status: Acknowledged