Audit Report

Archway Rewards Module

v1.0

June 5, 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 Archway Services Ltd. to perform a security audit of Archway's Rewards module.

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/archway-network/archway
Commit	`ec19e796aeef6ad90b313f20f4a3c296584aa52b`
Scope	Only the Rewards module in `x/rewards` and its integration into the Archway Cosmos SDK chain was in scope.

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

Archway is an incentivized smart contract platform that rewards developers for building on the network with baked-in incentives and rewards. This audit focused on the functionality associated with the rewards module. The rewards module consumes information gathered by tracking to create and distribute rewards.

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	**Low-Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**High**	The client provided detailed documentation, module specification, and a thorough litepaper.
Test coverage	**Medium-High**	The code in scope had extensive unit as well as end-to-end testing.

Summary of Findings

No	Description	Severity	Status
1	Non-deterministic iteration may break consensus	Critical	Resolved
2	Contract owners may set excessive `FlatFee`	Minor	Acknowledged
3	Missing usage description for transaction and query CLI commands	Informational	Acknowledged
4	Mismatch between technical specification and implementation	Informational	Resolved
5	Non-standard error codes	Informational	Resolved
6	Miscellaneous code quality comments	Informational	Partially Resolved

Detailed Findings

1. Non-deterministic iteration may break consensus

Severity: Critical

In a Cosmos SDK blockchain, non-determinism of blocks will cause the blockchain to halt. The iteration over the following maps is not deterministic: `blockDistrState.Txs`, `blockDistrState.Contracts`, and `contractDistrState.TxGasUsed` in `x/rewards/keeper/distribution.go:122, 133, 147` and `181` respectively. This can lead to consensus failure, since the order of iteration over a Go map is not guaranteed to be the same every time the program is executed.

Recommendation

We recommend sorting the order of `blockDistrState.Txs`, `blockDistrState.Contracts`, and `contractDistrState.TxGasUsed` before iterating over them, to ensure deterministic execution.

Status: Resolved

2. Contract owners may set excessive `FlatFee`

Severity: Minor

The `SetFlatFee` function in `x/rewards/keeper/msg_server.go:89` does not impose an upper-bound limit on the `FlatFee` amount. While the `FlatFee` is determined by the owner of a contract, governance should define a maximum fee to prevent excessive amounts that may impact the usability of contracts that are popular on the chain. While this is unlikely, contract owners may introduce prohibitively high fee amounts after their contract has achieved mass adoption.

Recommendation

We recommend implementing a maximum `FlatFee` amount that is controlled by governance.

Status: Acknowledged

The client acknowledges the finding and states that users and calling contracts can reject transactions that charge an excessive fee. The protocol allows flat fees to be queried, so this validation can be applied on external contracts that deem it necessary.

3. Missing usage description for transaction and query CLI commands

Severity: Informational

It is best practice to supply a long message for transaction and query CLI commands of a Cosmos SDK application, since such messages are helpful to both users and external developers. With the exception of `getTxSetContractMetadataCmd`, all transaction and query CLI commands for the rewards module in `x/rewards/client/cli/tx.go:24-28` and `x/rewards/client/cli/query.go:22-29` are missing long messages.

Recommendation

We recommend specifying a long message for all transaction and query CLI commands that describes how to correctly use the command.

Status: Acknowledged

4. Mismatch between technical specification and implementation

Severity: Informational

The technical specification in `x/rewards/spec/01_state.md:74` specifies that *"an Object is pruned (removed) at the BeginBlocker"*. However, objects are actually pruned in the `cleanupTracking` function in `x/rewards/keeper/distribution.go:237-238`, which is executed during the `EndBlocker`.

Recommendation

We recommend updating the technical documentation to reflect the current implementation.

Status: Resolved

5. Non-standard error codes

Severity: Informational

According to the official Cosmos SDK documentation (https://docs.cosmos.network/main/building-modules/errors), there are restrictions on error codes. One such restriction is that error codes must be greater than one, because the value one is reserved for internal errors. However, `ErrContractNotFound` in `x/rewards/types/errors.go:8` is registered with the error code one.

Recommendation

We recommend changing the error code of `ErrContractNotFound` to be greater than one, in line with the Cosmos SDK documentation.

Status: Resolved

6. Miscellaneous code quality comments

Severity: Informational

Throughout the codebase, some instances of inefficient code and misleading comments have been found.

Recommendation

The following are some recommendations to improve the overall code quality, efficiency and readability:

- Use `EmitTypedEvents` instead of `EmitEvents` in `x/rewards/ante/fee_deduction.go:88`.
- Remove the unused error `ErrContractFlatFeeNotFound` in `x/rewards/types/errors.go:12`.
- Handle the case where empty `rewards` are returned in `x/rewards/keeper/withdraw.go:29`, so that the function `withdrawRewardsByRecords` in line `32` is not called unnecessarily.
- Relocate the `metaUpdates.RewardsAddress` validation in `x/rewards/keeper/metadata.go:23-31` to be inside the conditional statement in lines `54-56`, to improve the code readability and efficiency.
- In the `deductFees` function in `x/rewards/ante/fee_deduction.go:110`, there is an incorrect comment that does not accurately represent the code it is describing. The comment mentions that if `hasWasmMsgs` is `false` it is set to `true`, but it is actually set to `hwm`.

Status: Partially Resolved