Audit Report

Coinhall Genie

v1.0

September 26, 2023

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.

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 Coinhall Genie CosmWasm smart 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/coinhall/genie-contracts
Commit	`957b308103d8285d0fae6d5f1cc365141c4f3f08`
Scope	All contracts were in scope.
Fixes verified at commit	`1fd141d5b98e4cc1127f405e1726037195220896`

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 Genie CosmWasm smart contract allows anyone to create an airdrop with configurable assets. Users can claim these assets by providing data that was signed with the configured public key.

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	-
Code readability and clarity	Medium-High	-
Level of documentation	Low-Medium	The codebase contained no inline documentation, but a README file including diagrams was provided.
Test coverage	Medium	There were JavaScript end-to-end tests, but no Rust unit tests. We recommend adding unit tests to the project.

Summary of Findings

No	Description	Severity	Status
1	Claiming may become impossible after many users have claimed	Major	Resolved
2	Not registering `campaign_id` may lead to unexpected issues in off-chain components	Minor	Resolved
3	Unbounded iteration over missions	Minor	Resolved
4	`from_timestamp` is not verified to be greater than the current time	Minor	Resolved
5	Additional funds sent to the contract are lost	Informational	Acknowledged
6	No handler for unexpected `cw20_msg.msg`, potentially leading to locked funds	Informational	Acknowledged
7	Suboptimal input data format for payout computation	Informational	Acknowledged
8	Contracts should implement a two step ownership transfer	Informational	Acknowledged
9	Inefficient claims payout computation	Informational	Acknowledged
10	“Migrate only if newer” pattern is not followed	Informational	Acknowledged
11	Unused `MigrateMsg` structure	Informational	Resolved
12	Airdrops are controlled by one private key	Informational	Acknowledged

Detailed Findings

1. Claiming may become impossible after many users have claimed

Severity: Major

The function `query_status` in the `contracts/genie-airdrop/src/state.rs` queries the number of elements of the `USERS` map with a `range` query. This operation is unbounded and may therefore eventually consume too much gas after many users have already claimed, leading to many entries in the `USERS` map. Because the function is called by the `handle_claim` and `handle_transfer_unclaimed_tokens` functions, this implies that it is no longer possible for users to claim and for the owner to retrieve unclaimed tokens. The funds would therefore be lost irrevocably.

Recommendation

We recommend using `is_empty` instead of counting the elements. Because it is only checked if the map has any entries (i.e. if the user count is non-zero), the number of elements is not needed.

Status: Resolved

2. Not registering `campaign_id` may lead to unexpected issues in off-chain components

Severity: Minor

The `CreateAirdrop` message takes `campaign_id` as a parameter, which has a `String` type. This `campaign_id` is not used anywhere in the contract logic and according to the technical documentation, it serves as metadata for off-chain components.

While this does not generate a security risk for the contract itself, the fact that it is not registered in a `genie-airdrop-factory` contract mapping can lead to a situation where there will be two or more `genie-airdrop` contracts with the same `campaign_id`. Such a collision may mislead users and off-chain components and may have unintended effects on them, for example, compromising the integrity of the campaign due to overwriting records with a new, just created campaign with the same ID.

Recommendation

We suggest implementing logic to verify whether a given `campaign_id` has already been used to create an airdrop. This can be done by using a mapping or another data format, which will add a record in the form of `campaign_id` by inserting it during each `CreateAirdrop` execution, while verifying that this `campaign_id` has not been used yet.

In addition, due to the use of the `String` data type, we recommend ensuring that server-side input validation is sufficient to protect against potential injection vulnerabilities, such as Cross Site Scripting or SQL Injection, as well as misleading identifiers that contain whitespace or non-printable characters.

Status: Resolved

3. Unbounded iteration over missions

Severity: Minor

In `contracts/genie-airdrop/src/state.rs:211-238` an iteration happens over the `State` structure's `unclaimed_amounts` field of type `Vec<Uint128>`. When the airdrop is configured with an extensive number of missions, this unbounded iteration can lead to high gas consumption, up to the point where the call leads to an out-of-gas error.

Recommendation

We recommend enforcing a cap on the mission count to prevent unbounded iteration during payouts.

Status: Resolved

4. `from_timestamp` is not verified to be greater than the current time

Severity: Minor

When creating a campaign, the user provides two time values: `from_timestamp` and `to_timestamp`. During the `instantiation` operation, it is verified whether `to_timestamp` is greater than `from_timestamp`, so that the campaign lasts at least one time unit.

However, there is no verification that the `from_timestamp` is equal to or greater than `env.block.time.to_seconds()`. As a consequence, it is possible that the campaign will never start, or that it will not be possible to add funds to the campaign because the status will be incorrect.

Recommendation

We recommend adding additional validation to the `from_timestamp` parameter to ensure it is equal to or greater than `env.block.time.to_seconds`.

Status: Resolved

5. Additional funds sent to the contract are lost

Severity: Informational

In `contracts/genie-airdrop/src/state.rs:167`, a check is performed that ensures that in the transaction there is at least one `Coin` with the expected `denom` field.

This validation does not ensure however that no other native tokens have been sent, and any such additional native tokens are not returned to the user, so they will be stuck in the contract forever.

While blockchains generally do not protect users from sending funds to wrong accounts, reverting extra funds increases the user experience.

Recommendation

We recommend checking that the transaction contains only the expected tokens and no additional ones.

Status: Acknowledged

6. No handler for unexpected `cw20_msg.msg`, potentially leading to locked funds

Severity: Informational

In `contracts/genie-airdrop/src/contract.rs:118`, there is a `match` statement for the `cw20_msg.msg`. The only supported `Cw20HookMsg` is `IncreaseIncentives`.

However, if the message is different, the smart contract does not provide any action for it, neither is an error returned. The execution of the function will then end silently, which in turn may lead to locked funds in the contract.

We classify this issue as informational since it only occurs when a user sends funds to the contract with a different, unsupported message. However, to improve the user experience and reduce the impact of user errors, reverting in such situations is recommended.

Recommendation

We suggest returning an error to revert the transaction if `cw20_msg.msg` does not have the expected value.

Status: Acknowledged

7. Suboptimal input data format for payout computation

Severity: Informational

In `contracts/genie-airdrop/src/contract.rs:201-205`, the `claims_amount` parameter of type `Binary` is decoded into a `Vec<Uint128>`. The vector's length is tied to the preset number of missions. Oversized vectors with sparse rewards introduce inefficiencies, leading to high gas costs:

1. Sparseness of the actual data causes redundant data sent in the smart contract call. This data undergoes encoding at the client, transmission, and a multi-step on-chain decoding process, which includes UTF-8 decoding and splitting by comma.
   For example, the string `"0,0,0,0,13,0,0,0,0,0,0,0,37,0,0,0,0"` contains 17 elements but only 2 are really used.

2. A multitude of zero values in `claim_amounts` causes unnecessary loop iterations when calculating the total payout (refer to lines `238-250`). In the aforementioned example, 15 unnecessary iterations are performed, in addition to 2 productive iterations.

Recommendation

We recommend using a more efficient format. For instance, mission numbers together with reward amounts could be accepted. Both would invoke a private function for the computation.

Status: Acknowledged

8. Contracts should implement a two step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

9. Inefficient claims payout computation

Severity: Informational

In `contracts/genie-airdrop/src/contract.rs:238`, a `for` loop iterates over a potentially extensive vector of claim amounts. During this iteration, payouts are computed, but edge cases when `amount` and `user_info.claim_amounts[i]` are equal or both are zero are not filtered out. Such cases lead to several unnecessary checked arithmetic operations with zero while not influencing the final payout.

Recommendation

We recommend checking edge cases in the iteration and skipping unproductive ones.

Status: Acknowledged

10. "Migrate only if newer" pattern is not followed

Severity: Informational

The `genie-airdrop-factory` contract is currently migrated without regard to his version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

Recommendation

We recommend following the migrate "only if newer" pattern defined in the CosmWasm documentation.

Status: Acknowledged

11. Unused `MigrateMsg` structure

Severity: Informational

In `packages/genie/src/airdrop.rs:72`, an unused `MigrateMsg` structure with an empty body is declared. The only `migrate` function is defined in the `genie-airdrop-factory` module which imports a different `MigrateMsg` definition from `packages/genie/src/factory.rs`.

Migration is not implemented for the `genie-airdrop` contract, so the `MigrateMsg` from `airdrop.rs` is not needed. Future developers or reviewers of the code might be confused by the presence of a `MigrateMsg` without a corresponding migrate function.

Recommendation

We recommend removing the unused structure. The presence of a `MigrateMsg` implies that there is some migration logic that might use it, even if the struct itself does not carry any data.

Status: Resolved

12. Airdrops are controlled by one private key

Severity: Informational

In `contracts/genie-airdrop-factory/src/contract.rs:125`, newly created airdrop campaigns are always instantiated with the same public key. The corresponding private key has the ability to authorize claims for every campaign. If compromised, that key can be used to drain all airdrop campaigns. It is therefore very important that the key is properly secured and processes are implemented to prevent any unauthorized access.

Recommendation

We recommend implementing and documenting strong security measures to protect the private key. Consider using hardware security modules and performing regular security audits for all infrastructure components that access the key. Above that, we recommend using a multi-signature wallet or governance module such that the compromise of one key does not affect all airdrop campaigns.

Status: Acknowledged