Audit Report

MANTRA Claimdrop

v1.0

October 30, 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 GmbH has been engaged by MANTRA Ventures Limited to perform a security audit of the MANTRA Claimdrop 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/MANTRA-Finance/claimdrop-contract
Commit	`89313f05b7dc70eaba7d8f2320c66fb39cd6232c`
Scope	All contracts were in scope.
Fixes verified at commit	`615e7d3bdf4c983f0898d85310d4c72baf99dd98` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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 MANTRA Claimdrop contract can be used to distribute tokens to a list of addresses. The contract uses a Merkle tree to store the list of addresses and their corresponding token amounts. The root of the Merkle tree is stored on the contract's campaign and is used to verify the validity of the proofs submitted by the recipients.

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**, **Partially Resolved**, 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 a thorough overview of the contract during the initial kick-off call and provided additional documentation.
Test coverage	**High**	`cargo tarpaulin` reports a test coverage of 93.93%, with 418/445 lines covered.

Summary of Findings

No	Description	Severity	Status
1	Unable to receive rounding error compensation when there are no other new claims	Minor	Resolved
2	Users can claim unvested funds if a linear vesting distribution type is incorrectly configured so that it ends after the campaign	Minor	Resolved
3	Merkle tree with multiple claims per address leads to incorrect claiming	Minor	Acknowledged
4	Inefficiency in fallback calculation	Informational	Resolved
5	Misleading parameter name	Informational	Resolved
6	Insufficient utilization of known invariants to simplify the code	Informational	Resolved
7	Missing event emission	Informational	Resolved
8	Unused code	Informational	Resolved

Detailed Findings

1. Unable to receive rounding error compensation when there are no other new claims

Severity: Minor

The `compute_claimable_amount` function that is called as part of the airdrop claiming mechanism calls the `get_compensation_for_rounding_errors` function in `src/helpers.rs:131-146` to determine the compensation for rounding errors that may occur when calculating the claim amount. This compensation is added to `new_claims`, which is a `HashMap` that stores the user's claimable amount for each distribution type from the current claim call.

However, if none of the distribution types result in a reward (e.g., due to them being fully claimed), `new_claims` will be empty, resulting in an error in lines `142-144` when retrieving the claim for the slot where the rounding error compensation has been placed into. Consequently, the message reverts and the rounding error compensation is not received by the user.

We classify this issue as minor as only a very small amount of funds, i.e., “dust” is not refunded to the user.

Recommendation

We recommend not erroring in the `compute_claimable_amount` function when `new_claims` does not contain a claim for the slot that the rounding error compensation is placed into. Instead, we suggest adding a new claim for the slot with the compensation amount to `new_claims` and returning it. The calling function can then aggregate the previous and new claims and store them in the contract.

Status: Resolved

2. Users can claim unvested funds if a linear vesting distribution type is incorrectly configured so that it ends after the campaign

Severity: Minor

Creating a new campaign calls the `validate_campaign_distribution` function that is implemented in `src/msg.rs:290-352` to validate the distribution types. However, it is not checked if the distribution types’ end time is after the campaign end time. This can lead to a situation where the campaign has ended, but the distribution types are still active.

Specifically, when determining the compensation for rounding errors in the `get_compensation_for_rounding_errors` function after the campaign has ended,

any non-fully claimed distribution types will be added to the compensation. Consequently, if distribution types are incorrectly configured, users can claim funds that are not vested yet.

We classify this issue as minor since only a misconfiguration by the contract owner can cause it.

Recommendation

We recommend validating the end times of the distribution types in the `validate_campaign_distribution` function to ensure that they end before the campaign ends.

Status: Resolved

3. Merkle tree with multiple claims per address leads to incorrect claiming

Severity: Minor

Previously claimed airdrops are kept track of in the `CLAIMS` storage map, defined in `src/state.rs:16`. This information is important to prevent users from claiming the same airdrop multiple times and to calculate vesting periods correctly.

However, if, for any reason, the constructed Merkle tree contains duplicate leaves where the same address has multiple airdrop claims, those separate claims will be incorrectly combined in `CLAIMS`. As a result, a user would not be able to claim multiple lump sum distributions, vesting periods are not calculated correctly, and users would not be able to claim the full amount across multiple Merkle tree leaves due to the maximum claim limit invariant in `src/commands.rs:227-230`.

We classify this issue as minor since only the contract owner can cause it.

Recommendation

We recommend storing previous claims separately in `CLAIMS` for each Merkle tree leaf (e.g., by using a hash of the Merkle tree leaf and the proofs) or, alternatively, documenting that the Merkle tree should only contain one claim per address.

Status: Acknowledged

4. Inefficiency in fallback calculation

Severity: Informational

In `src/helpers.rs`, the `calculate_claim_amount_for_distribution` function handles the case of linear distribution and checks the time of the last claim against the end of the distribution. If that is the case, `FALLBACK_TIME = 0` is used to calculate the claim.

However, this case also means that the full eligible amount has already been claimed before, and no further rewards are needed. This means that instead of computing additional rewards, the function could simply return zero, which would be more efficient.

Note that the `FALLBACK_TIME` constant is currently defined as `0`, so the contract behaves as expected. However, if the constant is redefined in the future, it may lead to a loss of funds for the contract. The `FALLBACK_TIME` constant is also used in line `228`, where it has no effect since it is ignored during matching to the pattern `(slot, _)` in line `223`.

Recommendation

We recommend removing the `FALLBACK_TIME` constant and simplifying the calculation of the claim amount.

Status: Resolved

5. Misleading parameter name

Severity: Informational

In `src/helpers.rs:45`, the `sender` parameter of the `validate_claim` function is declared. In the calling `claim` function, defined in `src/commands.rs`, this parameter is either set to explicitly provide the `receiver` address or defaults to `info.sender`.

Consequently, the parameter name is misleading and decreases the codebase's maintainability.

Recommendation

We recommend renaming the `sender` parameter to `receiver`.

Status: Resolved

6. Insufficient utilization of known invariants to simplify the code

Severity: Informational

In `src/helpers.rs`, invariants can be used to simplify the code:

1. The `compute_claimable_amount` function implements all its logic only for the case when the campaign has started. In the other case, default values are returned implicitly. However, the "not started" error would be clearer from a development and user experience perspective.
2. The `get_compensation_for_rounding_errors` function handles the case `total_claimed < total_claimable_amount` in lines `222-234`. However, in line `231`, the expression `total_claimable_amount.saturating_sub(total_claimed)` is used, which is unnecessary since the subtraction cannot overflow if the left value is greater than the right value.

Recommendation

We recommend utilizing known invariants to simplify the code.

Status: Resolved

7. Missing event emission

Severity: Informational

The instantiation of the campaign contract in `src/contract.rs:28` does not emit any event associated with initial contract parameters, which contain the owner's address. Due to this, the owner set to the contract is not immediately visible, e.g., by off-chain components.

Recommendation

We recommend emitting an event containing the owner passed in the `InstantiateMsg` message.

Status: Resolved

8. Unused code

Severity: Informational

The `compute_campaign_id` function in `src/helpers.rs:259` is never used. Excess code decreases code readability and maintainability.

Recommendation

We recommend removing unused code.

Status: Resolved