Security Audit Report

MANTRA Claimdrop Updates

v1.0

September 19, 2025

Table of Contents

License

Disclaimer

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 MANTRA Claimdrop Updates.

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-Chain/mantra-contracts-claimdrop
Commit	`f489de0a896faa02f50953d77df9dfa2c4140754`
Scope	The scope is restricted to the changes between commit `89313f05b7dc70eaba7d8f2320c66fb39cd6232c` and `f489de0a896faa02f50953d77df9dfa2c4140754` (tag `v2.0.0-rc7`).
Fixes verified at commit	`93c2378ae82604a10f87b0dd2962db139cc022ac`

	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 claimdrop contracts were updated and the airdrop campaigns and their design was revamped, with new functionalities such as blacklisting or address replacement were added.

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	**Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	-
Test coverage	**High**	94.10% coverage

Summary of Findings

No	Description	Severity	Status
1	Accidental fund transfers are possible when creating a campaign	Minor	Resolved
2	Placeholder addresses do not support all valid ENS domains	Minor	Resolved
3	`Lump Sum` distributions can be scheduled post-campaign	Minor	Resolved
4	The campaign owner can be blacklisted	Minor	Resolved
5	`close_campaign` refunds only the `reward_denom`	Minor	Resolved
6	Campaign end time is not checked	Informational	Acknowledged
7	Redundant storage operations decrease performance	Informational	Resolved
8	Redundant campaign parameter	Informational	Resolved
9	Inconsistency in old/new address validation	Informational	Resolved
10	Inconsistent blacklist behavior on remove vs replace	Informational	Resolved
11	Improper usage of saturated arithmetics	Informational	Resolved
12	Redundant if statements	Informational	Resolved
13	Invariant can be enforced	Informational	Resolved
14	`unwrap()` may panic	Informational	Resolved
15	Code duplicates	Informational	Resolved

Detailed Findings

1. Accidental fund transfers are possible when creating a campaign

Severity: Minor

In `src/commands.rs:35-58`, the `create_campaign` function is responsible for creating a new campaign, called as part of the `ManageCampaign` message and `CampaignAction::CreateCampaign` action.

However, contrary to the previous implementation, `info.funds` is not validated anymore to ensure that the correct tokens, matching the campaigns `reward_denom`, are sent along with the transaction. Instead, it is intended that the campaign is topped up via regular `BankMsg::Send` messages. Consequently, it is possible to accidentally send the wrong tokens when creating a campaign, which could lead to the funds being locked in the contract, as there is no mechanism to handle or refund them.

Recommendation

We recommend either preventing funds from being sent when creating a campaign by using `cw_utils::nonpayable(&info)`, or validating that only tokens with the correct denom are sent.

Status: Resolved

2. Placeholder addresses do not support all valid ENS domains

Severity: Minor

In `src/helpers.rs:271-301`, the `validate_address_placeholder` function validates strings as addresses that are not standard Cosmos addresses, such as Ethereum addresses (0x...) or ENS domains, which are expected to be used as placeholders for the actual claim recipient addresses in the allocation list.

However, the validation on lines `288-299` only permits ASCII alphanumeric characters and dots. This does not align with the latest ENS domain standards. For instance, ENSIP-15 introduced support for a wider range of characters, including emojis. As a result, valid ENS domains like "raffy🚴.eth" would be considered invalid.

This limitation prevents users with such ENS domains from being included in an airdrop campaign, as their addresses cannot be added to the allocation list.

Recommendation

We recommend updating the character validation in the `validate_address_placeholder` function to be compliant with the official ENS normalization standard. This will ensure that all valid ENS domains are supported.

Status: Resolved

3. Lump Sum distributions can be scheduled post-campaign

Severity: Minor

In `src/helpers.rs:17`, the function `validate_campaign_distribution` is called. This function, defined in the `mantra-claimdrop-std-1.1.3` crate, ensures that distributions are set to conclude either before or simultaneously with the campaign they are part of. In `src/msg.rs:385-431` of the crate, all campaign distributions are iterated and validated.

One such validation is related to the `end_time` and is defined on lines `414-431`. However, this validation is bypassed for Lump Sum distributions, under the presumption that they do not have an "end time." This assumption is documented with the comment on line `414`. The specific validation that is skipped due to this assumption is defined on lines `424-425` as `*end_time <= self.end_time`.

The idea that Lump Sum distributions do not have an "end time" is not actually correct. In fact, the start and end times of such distributions coincide, as all scheduled tokens become accessible precisely at the `start_time`. Thus, on line `386`, where the `start_time` and `end_time` variables are initialized with values from either the `LinearVesting` or `LumpSum` variants, the `end_time` for `LumpSum` distributions should be set to the `start_time`, rather than a `None` value. Such an adjustment would render the `end_time` validation applicable on lines `414-431` for `LumpSum` distributions.

The direct consequence is that Lump Sum distributions can be scheduled for payout post-campaign, presenting potential misuse opportunities. Moreover, other subtle issues could arise from the incorrect validation of Lump Sum distributions.

For example, in `src/helpers.rs:234-256`, the function `distribution_types_ended` is defined. It is used in the claim process to determine when to compute compensation for rounding errors. However, a corner case emerges with Lump Sum distributions concluding post-campaign. In this case, the compensation will not be computed in time. As a consequence, the function `distribution_types_ended` might not return `true`, despite the `campaign` having concluded.

The issue has a limited impact of token loss in dust amounts due to inactivity of the `get_compensation_for_rounding_errors` function, and it can only be triggered by improper campaign configuration, hence it is reported with "Minor" severity.

Recommendation

We recommend correcting the `validate_campaign_distribution` function by setting the `end_time` variable not only for the `LinearVesting` variant, but for `LumpSum` as well. Consider also renaming the `start_time` field of the `LumpSum` to a more appropriate term.

Status: Resolved

4. The campaign owner can be blacklisted

Severity: Minor

In `src/commands.rs:155-156`, the `claim` function ensures that the `receiver` is not blacklisted before proceeding to the claim processing. However, there is a possibility that the campaign `owner` is added into both blacklist and allocations mappings. In such a case, the only way for the `owner` to withdraw the allocated tokens is to close the campaign.

In order to trigger the issue, any one or two of the authorized entities have to execute `AddAllocations` and `BlacklistAddress` commands in any order. The `add_allocations` function does not check the recipient against the owner, as defined in `src/commands.rs:348-360`. Similarly, the `blacklist_address` function does not check the address against the owner either, as defined in `src/commands.rs:489-495`.

There can be up to `1000` authorized wallets able to perform the aforementioned operations, according to the `MAX_AUTHORIZED_WALLETS_BATCH_SIZE` constant declared in `src/msg.rs:17-18`. As a consequence, there is an increased chance of one of the authorized wallets adding allocation for the `owner` or blacklisting the `owner`. In order to trigger the issue, it is not required to execute both operations by the same wallet—one wallet can blacklist, and another wallet can allocate.

Recommendation

We recommend disabling both the ability to blacklist the `owner` and the ability to add allocation for the `owner`, by performing the extra validation of the addresses in the functions `blacklist_address` and `add_allocations`.

Status: Resolved

5. `close_campaign` refunds only the `reward_denom`

Severity: Minor

In `src/commands.rs:76`, function `close_campaign` refunds only the `reward_denom` by querying a single balance and sending it to the owner. This is incorrect because any non-reward denom accidentally sent to the contract remains after close, effectively locking those funds.

Recommendation

We recommend adding an owner-only `sweep(denom)` admin path to withdraw non-reward balances.

Status: Resolved

6. Campaign end time is not checked

Severity: Informational

In `src/commands.rs:104-298`, the `claim` function allows users to claim their allocated tokens. The function checks if the campaign has started and if it has been manually closed by the contract owner. However, it does not check if the campaign has naturally ended based on its `end_time` property.

The `Campaign` struct contains an `end_time` which is intended to mark the conclusion of the campaign. The associated helper function `has_ended` checks this `end_time` against the current block time. Because the `claim` function does not use `has_ended`, the `end_time` property is not enforced, allowing claims to continue indefinitely unless the campaign is manually closed. This might not be the intended behavior.

Recommendation

We recommend incorporating a check for the campaign's `end_time` in the `claim` function by using the `has_ended` helper function. Alternatively, if the `end_time` is not meant to be enforced for claims or when closing the campaign, consider removing the `end_time` property and the `has_ended` function to avoid confusion.

Status: Acknowledged

The client acknowledges this finding, noting that the campaign `end_time` is intentionally not checked. Users should be able to claim their allocated tokens even after the `end_time` has passed, as long as the campaign has not been manually closed by the owner.

7. Redundant storage operations decrease performance

Severity: Informational

Throughout the codebase, multiple storage operations could be optimized:

- In `src/helpers.rs:38`, the result of the call to `get_claims_for_address` function involves storage access to the `CLAIMS` map. This is performed within the call to `compute_claimable_amount` function from the claim function on line `168`. The same value is computed again on line `217`.
- In `src/commands.rs:289-291`, the call to `get_total_claims_amount_for_address` is done, which retrieves claims from

the `CLAIMS` map. This data is used to ensure that `total_user_allocation` is greater than the total claimed amount. However, the storage access is redundant since the same claims are stored in the `updated_claims` variable.

- In `src/state.rs:29`, the storage map `BLACKLIST` defines its values as of `bool` type. This is redundant since `false` values are never used. The map can use type `()` for the values.
- Multiple locations retrieve full values from the storage maps, instead of only querying the existence of the keys. The method `has` is significantly more efficient than `may_load`:
    - In `src/state.rs:99`, to check if the address is blacklisted.
    - In `src/state.rs:122` and `src/state.rs:148`, to check if the address is authorized.
    - In `src/commands.rs:352` and and `src/commands.rs:398`, to check if the address has no allocations.
    - In `src/commands.rs:412-413`, to check if the address has made no claims.

Recommendation

We recommend re-using known data, optimizing the storage types and adopting the method that checks the existence of a key without parsing or interpreting the attached value.

Status: Resolved

8. Redundant campaign parameter

Severity: Informational

In `src/msg.rs` of the crate `mantra-claimdrop-std-1.1.3`, the `CampaignParams` structure is defined with fields `reward_denom` of type `String` and `total_reward` of type `Coin`. However, `total_reward.denom` is the same as `reward_denom`. This is validated on line `478` by the function `validate_rewards`.

Since data of the field `reward_denom` is already stored in the field `total_reward`, this poses potential for mistakes during campaigns configuration. While such mistakes would be detected by the `validate_rewards` function, removing the root cause of them would improve the user experience.

Recommendation

We recommend removing the field `reward_denom` and implementing it as a simple auxiliary function that returns the `total_reward.denom` value.

Status: Resolved

9. Inconsistency in old/new address validation

Severity: Informational

In `src/commands.rs:378`, the old address is validated with `validate_raw_address` (allows placeholders such as ENS-like strings), while the new address uses `addr_validate` (must be a valid chain address). This is inconsistent; as a result, replacing an allocation is more strictly constrained than the original entry, which limits flexibility in protocol management.

Recommendation

We recommend choosing and enforcing one policy consistently: either allowing placeholders in old and new addresses, or allowing the old address to be only a placeholder and requiring the new address to be canonical only.

Status: Resolved

10. Inconsistent blacklist behavior on remove vs replace

Severity: Informational

In `src/commands.rs:457`, when removing an address, the allocation entry is deleted but any existing blacklist entry is left intact (address remains blacklisted).

When replacing an address, the old address is removed from the blacklist and the new address is marked blacklisted instead.

This asymmetry can be confusing and may not match policy expectations (e.g., whether blacklist should follow the identity or the allocation).

Recommendation

We recommend removing the blacklist entry along with removal of the user address from the protocol.

Status: Resolved

11. Improper usage of saturated arithmetics

Severity: Informational

Throughout the codebase, saturated arithmetic is used although in each specific case there could be a better solution:

- In `src/helpers.rs:225`, the expression `total_claimable_amount.saturating_sub(total_claimed)` is returned as the value of the compensation for rounding errors. However, if `total_claimable_amount` is less than `total_claimed` that must be treated as a broken assumption. Returning zero in such a case is not desired and would mask the invalid state.

- In `src/helpers.rs:146`, the distribution duration is defined as `end_time.saturating_sub(*start_time)`. However, if `end_time` is less than `start_time`, that must be an invalid distribution and should be validated during configuration. Implicitly using zero instead of error can lead to subtle issues.
- Similarly in `src/helpers.rs:157`, the time passed since the distribution start is defined as `current_time.seconds().saturating_sub(*start_time)`. However, if `current_time.seconds()` is less than `start_time` then we should simply skip this distribution and return early. That would optimize gas costs and also make the code simpler to analyze.
- In `src/commands.rs:252`, the `remaining_to_distribute` variable is updated using the `remaining_to_distribute.saturating_sub(take_from_slot)` expression. The variable `take_from_slot` is defined on line `248` as `std::cmp::min(remaining_to_distribute, *available_from_slot)`. That means that the saturation occurs when `available_from_slot` is less than `remaining_to_distribute`. However, in this case the clearer control flow would be to break immediately instead of saturating. If `remaining_to_distribute` is set to zero, the loop is stopped only on the next iteration, on line `243`.

While saturated arithmetic can be used to prevent overflows and underflows it also often reduces clarity of the code while providing a suboptimal protection due to its implicit nature.

Recommendation

We recommend replacing usages of `saturating_sub` with explicit assertions, `if` statements or `checked_sub`.

Status: Resolved

12. Redundant `if` statements

Severity: Informational

In `src/commands.rs:221-275`, the function claim does not utilize known invariants to the full extent. There are two if statements that miss the corresponding else clause which could be an issue if the if statements themselves were not redundant:

- In `src/commands.rs:221`, the `remaining_to_distribute` variable is checked for having a positive value. However, it is initialized on line `219` as `actual_claim_amount_coin.amount`, which is initialized and validated within lines `176-199`.
- In `src/commands.rs:247` and `src/commands.rs:263`, the `new_claims.get(&slot_idx)` expression is conditionally unwrapped. However, it is always the `Some` variant since the variable `slot_idx` iterates through `lump_sum_slots_with_new_claims` and

`linear_vesting_slots_with_new_claims` which contains indices from `new_claims` by the construction.

Redundant if statements decrease readability and clarity and increase difficulty of maintaining the code. In general, missing `else` clauses often indicate potential issues as well.

Recommendation

We recommend removing redundant `if` statements or implementing the missing `else` clauses. In case of complex assumptions involved, consider asserting them using the `ensure!` macro.

Status: Resolved

13. Invariant can be enforced

Severity: Informational

In `src/commands.rs:276-278`, the comment states that “At this point, if initial checks were correct ..., `remaining_to_distribute` should be zero.”

However, this invariant is not guaranteed.

Recommendation

We recommend enforcing the invariant using the `ensure!` macro.

Status: Resolved

14. `unwrap()` may panic

Severity: Informational

In `src/helpers.rs:212`, the `acc.checked_add(*amount).unwrap()` can panic on overflow, which is discouraged in on-chain code. Even if overflow is unlikely, panics reduce visibility and produce unclear errors.

Recommendation

We recommend returning a predefined error instead.

Status: Resolved

15. Code duplicates

Severity: Informational

In the codebase, several code duplicates have been discovered:

- In `src/commands.rs:221-275`, the function claim performs the claim processing on lines `240-255`. Then, the same logic is duplicated on lines `257-274`. The only differences between the duplicates are the vectors that are iterated. These vectors or their iterators could be simply concatenated.
- In `src/state.rs:112-127`, the function `is_authorized` is defined. Its definition is duplicated in the function `assert_authorized`, defined in `137-158`.

Duplicated code not only decreases clarity and maintainability, but also poses security risks because issue fixes need to be ported to all duplicates manually.

Recommendation

We recommend extracting auxiliary functions and preparing vectors for iterations in order to eliminate code duplicates.

Status: Resolved