Audit Report

DAO DAO Rewards Distributor, `x/onft` Staked Voting, BitSong FanToken Factory, Permission Granularity

v1.2

November 7, 2024

Table of Contents

Table of Contents **2**
License **3**
Disclaimer **4**
Introduction **5**
Purpose of This Report 5
Codebase Submitted for the Audit 5
Methodology 6
Functionality Overview 6
How to Read This Report **7**
Code Quality Criteria **8**
Summary of Findings **9**
Detailed Findings **10**
1. Rewards are lost if the distribution is funded after the expiry 10
2. Incorrect reward distribution due to inconsistent voting power queries 10
3. Users can inflate their voting power by registering duplicate NFT token IDs 11
4. Rounding error during voting power updates causes undistributed rewards 11
5. Missing validation of submission policy during contract migration 12
6. Units of duration are not distinguished 12
7. Expired linear distributions cannot be paused 13
8. Suboptimal storage update 13
9. Duplicate distribution configurations can be created 14
10. Suboptimal usage of optional vectors 14
11. Storage collection considerations 15
12. Code deduplication 16
13. Unnecessary code 16
14. “Migrate only if newer” pattern is not followed 16
Appendix **18**
1. Test case for “Rewards are lost if the distribution is funded after the expiry” 18
2. Test case for “Incorrect reward distribution due to inconsistent voting power queries” 19

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 Intentional Design LLC to perform a security audit of DAO DAO Rewards Distributor, `x/onft` Staked Voting, BitSong FanToken Factory, Permission Granularity.

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/DA0-DA0/dao-contracts
Commit	`75272b92867e057df108f5961eb34dd179e75dea`
Scope	The scope was restricted to the changes compared to commit `3ab701747d6d5571cfee7f5a78d8fe39ffa3a8f7`.
Fixes verified at commit	`05c982be70a2a3c80b34f164af2e5f894b61b19b` 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 submitted code implements the smart contracts for DAO DAO, which creates a modular framework for creating DAOs, including staking and voting functionalities.

The code within this audit's scope introduces new functionalities, such as a rewards distributor, `x/onft` staked voting, BitSong FanToken factory contracts, and enhancements in permission granularity.

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-High	Complex reward distribution and transition logic are implemented.
Code readability and clarity	Medium-High	Most functions are well-documented with concise and clear code comments.
Level of documentation	High	Detailed documentation was available at https://github.com/DA0-DA0/dao-contracts/wiki/DAO-DAO-Contracts-Design and in `README` files.
Test coverage	Medium-High	`cargo-llvm-cov` reports the following coverage: • `87.84%` function coverage • `94.41%` line coverage • `75.64%` region coverage

Summary of Findings

No	Description	Severity	Status
1	Rewards are lost if the distribution is funded after the expiry	Critical	Resolved
2	Incorrect reward distribution due to inconsistent voting power queries	Critical	Resolved
3	Users can inflate their voting power by registering duplicate NFT token IDs	Critical	Resolved
4	Rounding error during voting power updates causes undistributed rewards	Major	Resolved
5	Missing validation of submission policy during contract migration	Minor	Resolved
6	Units of duration are not distinguished	Minor	Resolved
7	Expired linear distributions cannot be paused	Minor	Resolved
8	Suboptimal storage update	Informational	Resolved
9	Duplicate distribution configurations can be created	Informational	Acknowledged
10	Suboptimal usage of optional vectors	Informational	Resolved
11	Storage collection considerations	Informational	Acknowledged
12	Code deduplication	Informational	Resolved
13	Unnecessary code	Informational	Resolved
14	“Migrate only if newer” pattern is not followed	Informational	Resolved

Detailed Findings

1. Rewards are lost if the distribution is funded after the expiry

Severity: Critical

In `contracts/distribution/dao-rewards-distributor/src/contract.rs:303`, the `execute_fund` function restarts the distribution if it is not continuous and already expired at the current block. This is problematic because rewards accrued from the last updated timestamp `active_epoch.last_updated_total_earned_puvp` to the ending timestamp `active_epoch.ends_at` will be lost.

Consequently, stakers will not receive the rewards for the staking period, causing a loss of rewards.

Please see the `test_lost_rewards` test case in the appendix to reproduce this issue.

Recommendation

We recommend accruing rewards for stakers before restarting the distribution.

Status: Resolved

2. Incorrect reward distribution due to inconsistent voting power queries

Severity: Critical

In `contracts/distribution/dao-rewards-distributor/src/rewards.rs:113-136`, the `get_active_total_earned_puvp` function computes the reward index by dividing the accrued rewards by the total voting power queried from the **previous** block height (see `contracts/distribution/dao-rewards-distributor/src/helpers.rs:19`). This is incorrect because the user reward is computed with voting power queried from the **current** block height, as seen in `contracts/distribution/dao-rewards-distributor/src/helpers.rs:33`.

Consequently, the reward index will be computed using an outdated total voting power, resulting in incorrect reward distribution. An attacker can exploit this issue by staking many tokens one block before claiming the rewards to receive more at the expense of other users.

Please see the `test_claim_more_rewards` test case in the appendix to reproduce this issue.

Recommendation

We recommend computing the reward index with the total voting power queried from the current block height.

Status: Resolved

3. Users can inflate their voting power by registering duplicate NFT token IDs

Severity: Critical

In `contracts/voting/dao-voting-onft-staked/src/contract.rs:149`, the `execute_confirm_stake` function does not validate that the supplied NFT token IDs are not duplicates. This is problematic because the `register_staked_nfts` function in `contracts/voting/dao-voting-onft-staked/src/state.rs:65-77` increases the user's voting power based on the total number of token IDs supplied, which may include duplicates.

An attacker can exploit this issue by calling the `execute_confirm_stake` function with duplicates of a token ID, granting them a high amount of voting power without requiring them to stake the necessary NFTs. These voting powers can be weaponized to manipulate the DAO into dispatching malicious messages, such as transferring funds to the attacker.

Recommendation

We recommend returning an error in the `execute_confirm_stake` function if the supplied NFT token IDs contain duplicates.

Status: Resolved

4. Rounding error during voting power updates causes undistributed rewards

Severity: Major

In `contracts/distribution/dao-rewards-distributor/src/rewards.rs:121-130`, the `get_active_total_earned_puvp` function computes the completed distribution periods by dividing the `new_reward_distribution_duration` with `duration`. The division is implemented as `checked_div`, which returns the variable as an unsigned integer.

This is problematic because a rounding error will occur if the `duration` is larger than `new_reward_distribution_duration`, causing the `complete_distribution_periods` to zero. This scenario could happen when a voting

power change occurs before a full interval has elapsed, resulting in accrued rewards being undistributed and stuck in the contract.

This issue has been independently reported by the client.

Recommendation

We recommend computing the `new_reward_distribution_duration` as `Decimal`.

Status: Resolved

5. Missing validation of submission policy during contract migration

Severity: Minor

In `packages/dao-pre-propose-base/src/execute.rs:670`, the `migrate` function allows the contract migration admin to set the `PreProposeSubmissionPolicy` when updating the contract. However, no validation ensures the supplied policy does not accidentally lock the DAO due to an invalid configuration.

We classify this issue as minor because it can only be caused by the contract migration admin, which is a privileged account.

Recommendation

We recommend validating the policy during the migration, similar to `packages/dao-pre-propose-base/src/execute.rs:67`.

Status: Resolved

6. Units of duration are not distinguished

Severity: Minor

Time durations are crucial to the logic of the `dao-rewards-distributor` contract. However, the data types implemented are error-prone.

For example, the `get_exp_diff` function in `contracts/distribution/dao-rewards-distributor/src/helpers.rs:78` subtracts one `Expiration` from another. Although the `end` and `start` parameters explicitly indicate the unit duration types (i.e., blocks or seconds), this information is not preserved in the output as the function returns `u64` instead of the `Expiration` struct.

Additionally, the `get_exp_diff` function returns `0` if the input parameters are both `Expiration::Never`, which is mathematically incorrect.

Similarly, the `get_duration_scalar` function in line `42` removes the duration unit information and directly returns the inner value as `u64`, further elevating the potential for errors.

Consequently, the code responsible for managing distribution schedules and timeframes becomes more challenging to interpret, particularly when it involves arithmetic operations with mixed time units. This complexity increases the likelihood of bugs in these segments.

Recommendation

We recommend employing Rust traits for time arithmetic to prevent issues related to mixing different units of time.

Status: Resolved

7. Expired linear distributions cannot be paused

Severity: Minor

In `contracts/distribution/dao-rewards-distributor/src/state.rs:271-272`, the `transition_epoch` function sets the distribution ending period (`Epoch.ends_at`) to the current block height or timestamp. This is problematic because the distribution may have already expired in the previous block, which causes the ending period to be updated incorrectly.

Consequently, expired linear distributions cannot be paused due to an overflow error in `contracts/distribution/dao-rewards-distributor/src/state.rs:285-288`, preventing the protocol from working as intended.

This issue has been independently reported by the client.

Recommendation

We recommend using the `get_latest_reward_distribution_time` function to determine the distribution ending period.

Status: Resolved

8. Suboptimal storage update

Severity: Informational

In `packages/dao-pre-propose-base/src/execute.rs:199`, the `execute_update_config` function validates the `submission_policy` parameter in line `213` and updates it in the `Config` state.

However, this is inefficient because the `Config` state is retrieved in line `211` before validating the submission policy. If the submission policy does not pass the validation, the loaded storage value (indicated as the `prev` variable) remains unused.

Recommendation

We recommend validating the `submission_policy` parameter before retrieving the `Config` state to avoid an unnecessary storage read.

Status: Resolved

9. Duplicate distribution configurations can be created

Severity: Informational

In `contracts/distribution/dao-rewards-distributor/src/contract.rs:129`, the `execute_create` function allows the contract owner to create a distribution. However, other distributions may already exist with the same configuration, causing duplicate distributions to be created.

Recommendation

We recommend hashing the distributions and recording them in a storage map to ensure duplicate configurations cannot be created.

Status: Acknowledged

The client states that they are okay with this as users are able to update emission rates on a per-distribution basis, and thus there is a possibility users may want distribution configs to be the same at some points and different at other points. Also, the front end can abstract over all active distributions, so this is invisible to the reward recipients, and the client can enforce best practices when creating new distributions.

10. Suboptimal usage of optional vectors

Severity: Informational

The `Option<Vec<...>>` type is prevalent across the codebase. Although it is useful to distinguish between an empty and an absent list in certain scenarios, this distinction is currently not used in the codebase, which is inefficient and increases code complexity.

For example, in `packages/dao-voting/src/pre_propose.rs:72-84`, the `PreProposeSubmissionPolicy` enum declares the `denylist` and `allowlist` fields to `Option<Vec<String>>`. The lists are parsed with `unwrap_or_default`, which converts empty lists to `None` before updating the submission policy configuration in

`packages/dao-pre-propose-base/src/execute.rs:291` and `365`. This indicates there is no meaningful distinction between `None` and an empty vector.

Hence, we conclude that `Option<Vec<...>>` serves more as an optimization than a necessity. Given that modern Rust minimizes the memory footprint of empty vectors, the complexity added by `Option<Vec<...>>` pattern outweighs its benefits.

Consequently, the code complexity is increased due to additional checks and unwrapping without a significant memory efficiency gain.

Recommendation

We recommend using plain vectors when there is no requirement to distinguish between an empty and absent collection.

Status: Resolved

11. Storage collection considerations

Severity: Informational

In `contracts/voting/dao-voting-onft-staked/src/state.rs:33`, the `STAKED_NFTS_PER_OWNER` state is defined as `Map<(&Addr, &str), Empty>`. This mapping type is optimal for scenarios when tokens are added incrementally, so the storage updates only mutate the affected entries without needing to read and write other unaffected tokens.

However, `STAKED_NFTS_PER_OWNER` can also be used in batch updates in lines `72-75` or `101-110`. This scenario is problematic in the current implementation because it requires querying and mutating multiple storage entries one by one, increasing the gas consumption.

Recommendation

We recommend using `Map<Addr, HashSet<String>>` for the storage map.

Status: Acknowledged

The client states that while there are some benefits to storing staked NFTs in a `HashSet`, they lose the ability to utilize `Map` prefix pagination when listing staked NFTs and are forced to load the entire `HashSet` when they want to access any individual one. This optimization trade-off does not make sense to them, and if the `HashSet` is too large, it will pose gas issues. Additionally, the client uses the `Map` to `Empty` pattern in other audited contracts, so this keeps the patterns consistent across the codebase.

12. Code deduplication

Severity: Informational

The code segments in `packages/dao-pre-propose-base/src/execute.rs:262-283`, `307-328`, and `330-351` are identical. These segments implement modifying permission lists with iteration, deduplication, and address validation, which can be abstracted as reusable components to increase code maintainability.

Recommendation

We recommend defining a function that implements the core logic and reusing it across the codebase.

Status: Resolved

13. Unnecessary code

Severity: Informational

In `contracts/voting/dao-voting-onft-staked/src/contract.rs:178-180`, the `execute_confirm_stake` function removes the `PREPARED_ONFTS` state based on the supplied token IDs. This is unnecessary because the `register_staked_nfts` function already removes them in `contracts/voting/dao-voting-onft-staked/src/state.rs:73`.

Recommendation

We recommend removing unnecessary code.

Status: Resolved

14. “Migrate only if newer” pattern is not followed

Severity: Informational

In `contracts/distribution/dao-rewards-distributor/src/contract.rs:594-597`, the rewards distributor contract can be migrated without regard to their version. This can be improved by adding validation to ensure that 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: Resolved

Appendix

1. **Test case for "Rewards are lost if the distribution is funded after the expiry"**

Please run the test case in `contracts/distribution/dao-rewards-distributor/src/testing/tests.rs`.

```
#[test]
fn test_lost_rewards() {
       let mut suite = SuiteBuilder::base(super::suite::DaoType::Native)
       .with_rewards_config(RewardsConfig {
       amount: 3_000,
       denom: UncheckedDenom::Native(DENOM.to_string()),
       duration: Duration::Height(1),
       destination: None,
       continuous: false,
       })
       .build();

       suite.assert_amount(3_000);
       suite.assert_ends_at(Expiration::AtHeight(33_333));
       suite.assert_duration(1);

       // skip to end
       suite.skip_blocks(33_333);

       // check pending rewards
       suite.assert_pending_rewards(ADDR1, 1, 49999500);
       suite.assert_pending_rewards(ADDR2, 1, 24999750);
       suite.assert_pending_rewards(ADDR3, 1, 24999750);

       // before user claim rewards, someone funded
       suite.fund_native(1, coin(1u128, DENOM));

       // pending rewards are now gone
       suite.assert_pending_rewards(ADDR1, 1, 0);
       suite.assert_pending_rewards(ADDR2, 1, 0);
       suite.assert_pending_rewards(ADDR3, 1, 0);

       // no rewards claimable
       suite.claim_rewards(ADDR1, 1);
}
```

2. Test case for "Incorrect reward distribution due to inconsistent voting power queries"

Please run the test case in `contracts/distribution/dao-rewards-distributor/src/testing/tests.rs`.

```
#[test]
fn test_claim_more_rewards() {
       let mut suite = SuiteBuilder::base(super::suite::DaoType::Native)
       .with_rewards_config(RewardsConfig {
       amount: 3_000,
       denom: UncheckedDenom::Native(DENOM.to_string()),
       duration: Duration::Height(1),
       destination: None,
       continuous: true,
       })
       .build();

       suite.assert_amount(3_000);
       suite.assert_ends_at(Expiration::AtHeight(33_333));
       suite.assert_duration(1);

       // ADDR1 stake big amount of tokens
       suite.skip_blocks(33_000);
       suite.mint_native(coin(10_000, &suite.reward_denom), ADDR1);
       suite.stake_native_tokens(ADDR1, 10_000);

       // ADDR1 claims rewards in the next block
       suite.skip_blocks(1);
       suite.claim_rewards(ADDR1, 1);

       // skip to end
       suite.skip_blocks(100_000_000);

       // all users claim rewards
       suite.claim_rewards(ADDR1, 1);
       suite.claim_rewards(ADDR2, 1);
       suite.claim_rewards(ADDR3, 1); // last user cannot claim rewards due to
insufficient balance
}
```