Audit Report

DAO DAO

v1.0

June 22, 2022

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 In With The New to perform a security audit of the DAO DAO 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 GitHub repository:

https://github.com/DA0-DA0/dao-contracts

Commit hash: `f802e7932b1587fd4c55c530c09c3fdfb8253666`

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.

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.

Summary of Findings

No	Description	Severity	Status
1	Duplicate member addresses inflate the total weight value	Major	Resolved
2	Duplicate item name causes ghost contracts to be instantiated	Major	Resolved
3	Lack of delay when executing proposals makes contracts prone to governance attacks	Major	Resolved
4	Consider checking whether the item key exists before removing	Minor	Resolved
5	Admin withdrawals will affect the reward distribution	Minor	Acknowledged
6	Reward duration should be validated as non-zero value	Minor	Resolved
7	Maximum number of items is not sufficient to prevent out of gas error	Minor	Resolved
8	`execute_pause` does not enforce maximum pause duration	Minor	Acknowledged
9	Incomplete threshold validation in staked balance voting contract	Minor	Resolved
10	Admin rights go against best practices	Informational	Acknowledged
11	Voting module design introduces configuration risk	Informational	Acknowledged
12	Contracts should implement a two-step ownership transfer	Informational	Resolved
13	Additional funds sent to the contract are lost	Informational	Resolved
14	Voting thresholds can be set to unusual values	Informational	Resolved
15	Overflow checks not enabled for release profile	Informational	Acknowledged
16	Outstanding `TODO` comments are present in the codebase	Informational	Resolved
17	Messages with zero rewards are inefficient	Informational	Resolved

18	Saving zero-valued voting weights is inefficient	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	-
Test coverage	Medium-High	-

Detailed Findings

1. Duplicate member addresses inflate the total weight value

Severity: Major

During the contract instantiation phase in `contracts/cw4-voting/src/contract.rs:32-37`, the `msg.initial_members` vector that contains the `Member` struct address and weight is not validated to make sure the member address is unique across the whole contract. If a duplicate member address is provided, the associated address and weight would be overwritten by the `USER_WEIGHTS` storage state, as seen in line `35`. However, the total weight value in line `36` would still include the nonexisting member weight. This would impact voting outcomes.

A test case demonstrating the above scenario can be found in appendix 1.

Recommendation

We recommend deduplicating member addresses in the `msg.initial_members` vector.

Status: Resolved

2. Duplicate item name causes ghost contracts to be instantiated

Severity: Major

In `contracts/cw-core/src/contract.rs:83`, the `msg.initial_items` parameter that contains the `InitialItem` vector is not validated to have unique item names. Having two or more items with the same name would cause them to be instantiated as "ghost contracts" because eventually, the item name with the highest reply id would be used, as seen in line `746`. As a result, this causes inefficiencies in the contract.

Recommendation

We recommend deduplicating item names in the `msg.initial_items` vector.

Status: Resolved

3. Lack of delay when executing proposals makes contracts prone to governance attacks

Severity: Major

In `contracts/cw-proposal-single/src/contract.rs:217`, there are no time delays when executing passed proposals. An attacker with enough capital can potentially

stake many tokens to gain high voting power, create a malicious proposal and then vote for it. Since proposals can pass early as long as the voting threshold is reached (see `contracts/cw-proposal-single/src/proposal.rs:112-114`) to execute a passed malicious proposal without giving the community enough time to discuss and block the governance attack. As a result, the attacker's governance attack would succeed, causing potentially catastrophic damage to the protocol. This is especially problematic if the contracts hold non-native tokens, which may have a higher value than the cost of the attack.

Recommendation:

We recommend adding a minimum time delay that must pass before proposals can be executed.

Status: Resolved

4. Consider checking whether the item key exists before removing

Severity: Minor

In `contracts/cw-core/src/contract.rs:420-434`, any provided `key` as an argument will be removed from the `ITEMS` storage state as seen in line `430`. Since the `remove` storage function will not differentiate whether the key exists or not, it is possible to remove a key that never existed in the storage.

Recommendation

We recommend verifying the key exists before removing them using the `has` storage function.

Status: Resolved

5. Admin withdrawals will affect the reward distribution

Severity: Minor

The `execute_withdraw` function in `contracts/stake-cw20-reward-distributor/src/contract.rs:185` allows the admin to withdraw the entire balance of the contract without an option for a partial withdrawal.

After the admin withdraws funds using a `Withdraw` message, the contract will have empty funds, which causes the minimum comparison in line `156` to result in a zero amount. This implies that the `Distribute` message would pay zero rewards to the staking address for the whole staked duration. The staking address would need to wait some time for the pending rewards to recover to the intended value to receive another set of staking rewards.

We consider this to be a minor issue since only the contract owner can cause it.

Recommendation

We recommend adding support for partial withdrawals.

Status: Acknowledged

The DAO DAO team stated that this functionality will only be used in emergencies.

6. Reward duration should be validated as non-zero value

Severity: Minor

In `contracts/stake-cw20-external-rewards/src/contract.rs:65`, `msg.reward_duration` represents the reward duration value which is used to calculate the reward rate. If the reward duration value provided is `0`, it would cause a division by zero error in line `174` during the `execute_fund` functionality, causing the contract to be unable to be funded.

We consider this to be a minor issue since only the contract owner can cause it. Even if it happens, the problem can be resolved by updating the reward duration to a non-zero value via the `UpdateRewardDuration` message.

Recommendation

We recommend validating that the reward duration value is not zero in lines `65` and `372`.

Status: Resolved

7. Maximum number of items is not sufficient to prevent out of gas error

Severity: Minor

The `MAX_ITEM_INSTANTIATIONS_ON_INSTANTIATE` validation in `contracts/cw-core/src/contract.rs:84` is not sufficient as it will allow for such a large number of items that the proceeding instantiation loop will run out of gas well before the limit is met. The maximum should be set to a more conservative value rather than `u64::MAX - 100`.

Recommendation

We recommend updating the `MAX_ITEM_INSTANTIATIONS_ON_INSTANTIATE` value with a more conservative value to prevent out of gas errors when instantiating new items.

Status: Resolved

8. `execute_pause` does not enforce maximum pause duration

Severity: Minor

The `execute_pause` function in `contracts/cw-core/src/contract.rs:173` does not enforce a maximum pause duration. The value of `pause_duration` is added to the current block time and then saved to `PAUSED` with no checks or validation. This is problematic because if the value is accidentally set to a very high value, it will effectively render the contract useless, without a way to recover. Another potential scenario is that the duration is sent as a `Height` rather than a `Time`, which may result in the contract being paused for longer than anticipated.

Recommendation

We recommend adding a maximum pause duration to check for both a maximum `Height` and `Time` depending on the type of `Duration` specified.

Status: Acknowledged

The DAO DAO team states that allowing large pause durations is intentional. This allows DAOs to effectively and permanently lock themselves.

9. Incomplete threshold validation in staked balance voting contract

Severity: Minor

In `contracts/cw20-staked-balance-voting/src/contract.rs:84`, `ActiveThreshold::Percentage` is only validated to be smaller than `100%`, but not validated to be larger than `0%`. Zero values may cause undesired voting outcomes.

Recommendation

We recommend validating the `ActiveThreshold::Percentage` to be larger than `0`.

Status: Resolved

10. Admin rights go against best practices

Severity: Informational

It is best practice to restrict admin permissions to actions that do not directly allow access to or could create a loss of user funds. In `contracts/cw-core/src/contract.rs:194`, the contract admin can run any message, introducing a centralization risk and a single point of failure if the admin keys are lost or compromised.

In other reports, centralization risks have been flagged with higher severity. In this case, we interpret the admin as a parent DAO which mitigates the centralization risk.

Recommendation:

We recommend adding a check to ensure that the admin is the parent DAO smart contract by performing a query message as seen in `contracts/stake-cw20-reward-distributor/src/contract.rs:27` and encouraging users to follow best practices by using a DAO as the admin of the protocol.

Status: Acknowledged

11. Voting module design is prone to configuration risk

Severity: Informational

The core contract and the voting module are closely interlinked. Because the voting module has only one interface (i.e. to the core contract), updating the voting module introduces an additional point of failure and a configuration risk (e.g. that a proposal module is configured as the voting module) without many benefits that come from the modular design.

Because this issue is based on a design choice the severity is informational.

Recommendation

We recommend integrating the voting module into the core contract.

Status: Acknowledged

The DAO DAO team states that the separate module is an intentional design choice with the following benefits:

- Reduces the code complexity of writing a new voting module as it can be completely separated from the existing core contract.
- Reduces the complexity of doing a security upgrade via a contract migration or module upgrade as it is easy to set a module to its admin, but setting the core contract as its admin is not possible unless a factory contract is used during instantiation that instantiates the core contract with itself as the admin and then transfers those admin rights. They will be implementing a factory contract like this down the line.
- Allows DAOs to upgrade to a new voting system without too much trouble. For example, a multisig style DAO may eventually grow large enough that it would like to switch to token-based voting, or a DAO may want to eventually add a secondary voting token that has different voting rights (e.g. an NFT collection gets launched which should be used for voting).

12. 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.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

13. Additional funds sent to the contract are lost

Severity: Informational

In `contracts/stake-cw20-external-rewards/src/contract.rs:143-153`, a check is performed that ensures that in the transaction there is a `Coin` with the expected `denom` field.

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

Recommendation

We recommend checking that the transaction contains only the expected `Coin` using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Resolved

14. Voting thresholds can be set to unusual values

Severity: Informational

In `packages/voting/src/threshold.rs:73-80`, it is possible to set `PercentageThreshold` such that a majority can be achieved under very unusual conditions (e.g. if `1%` are in favor and `99%` percent are against a proposal). While this allows

for innovative forms of governance, it is prone to user error which may lead to devastating consequences.

Recommendation

We recommend displaying warnings in the user interface if `PercentageThreshold` is set to a low value (e.g. below `50%`) and clearly documenting that setting the variable to very low values can lead to counterintuitive outcomes.

Status: Resolved

15. Overflow checks not enabled for release profile

Severity: Informational

The following packages and contracts do not enable `overflow-checks` for the release profile:

- `contracts/cw-core/Cargo.toml`
- `contracts/cw-proposal-single/Cargo.toml`
- `contracts/cw4-voting/Cargo.toml`
- `contracts/cw20-staked-balance-voting/Cargo.toml`
- `contracts/stake-cw20/Cargo.toml`
- `contracts/stake-cw20-external-rewards/Cargo.toml`
- `contracts/stake-cw20-reward-distributor/Cargo.toml`

While enabled implicitly through the workspace manifest, a future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Acknowledged

The DAO DAO team added comments in the workspace file detailing the importance of enabling integer overflow checks.

16. Outstanding `TODO` comments are present in the codebase

Severity: Informational

During the audit engagement, a `TODO` comment was found in `contracts/stake-cw20-external-rewards/src/contract.rs:839`. This implies that the contract might still be under development and not yet ready for mainnet deployment.

Recommendation

We recommend resolving the `TODO` issues and removing them from the codebase.

Status: Resolved

17. Messages with zero rewards are inefficient

Severity: Informational

In `contracts/stake-cw20-reward-distributor/src/contract.rs:146`, `block_diff` can be zero. That would trigger messages with no rewards, which is inefficient.

Recommendation

We recommend reverting with an error when `block_diff` is zero.

Status: Resolved

18. Saving zero-valued voting weights is inefficient

Severity: Informational

In `contracts/cw4-voting/src/contract.rs:33-35`, zero-valued voting weights may be saved, which is inefficient.

Recommendation

We recommend saving only non-zero weights.

Status: Resolved

Appendix

Test case for issue 1

```
#[test]
fn test_duplicate_member() {
    // NOTE: reproduced in contracts/cw4-voting/src/tests.rs file
    let mut app = App::default();
    let _voting_addr = setup_test_case(&mut app);

    let voting_id = app.store_code(voting_contract());
    let cw4_id = app.store_code(cw4_contract());

    // Instantiate with members but no weight
    let msg = InstantiateMsg {
        cw4_group_code_id: cw4_id,
        initial_members: vec![
            cw4::Member {
                addr: ADDR1.to_string(),
                weight: 25,
            },
            cw4::Member {
                addr: ADDR2.to_string(),
                weight: 25,
            },
            cw4::Member {
                addr: ADDR3.to_string(),
                weight: 25,
            },
            cw4::Member {
                addr: ADDR3.to_string(), // same address above
                weight: 25,
            },
        ],
    };
    let voting_addr = app
        .instantiate_contract(
            voting_id,
            Addr::unchecked(DAO_ADDR),
            &msg,
            &[],
            "voting module",
            None,
        )
        .unwrap();

    app.update_block(next_block);

    let total_voting_power: TotalPowerAtHeightResponse = app
        .wrap()
        .query_wasm_smart(
            voting_addr.clone(),
```

```
            &QueryMsg::TotalPowerAtHeight { height: None },
        )
        .unwrap();

    assert_eq!(total_voting_power.power, Uint128::from(100_u64)) // should be 75
due to duplicate address
}
```