Audit Report

DAO DAO 2

v1.0

February 6, 2023

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 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 GitHub repository:

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

Commit hash: `74bd3881fdd86829e5e8b132b9952dd64f2d0737`

The scope of this audit was limited to:

- `contracts/dao-core`
- `contracts/external/cw-admin-factory`
- `contracts/external/cw-token-swap`
- `contracts/pre-propose/dao-pre-propose-approval-single`
- `contracts/pre-propose/dao-pre-propose-approver`
- `contracts/pre-propose/dao-pre-propose-multiple`
- `contracts/pre-propose/dao-pre-propose-single`

- contracts/proposal/dao-proposal-multiple
- contracts/proposal/dao-proposal-single
- contracts/staking/cw20-stake
- contracts/staking/cw20-stake-external-rewards
- contracts/staking/cw20-stake-reward-distributor
- contracts/voting/dao-voting-cw4
- contracts/voting/dao-voting-cw20-staked
- contracts/voting/dao-voting-cw721-staked
- contracts/voting/dao-voting-native-staked
- contracts/voting/dao-voting-staking-denom-staked
- Relevant files in the packages/* folder

Additionally, the client implemented a Condorcet proposal contract in contracts/proposal/dao-proposal-condorcet as a fix for issue 11 of this report, which was included in this audit on commit 490a9e8eb1704d0207d03286d065693b9e17fa85.

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.

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**	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 `README` files.
Test coverage	**Medium-High**	-

Summary of Findings

No	Description	Severity	Status
1	Queried voting power when executing proposals is set to current height, allowing anyone to execute proposals	Major	Resolved
2	Users might be unable to claim NFTs if they unstake a large amount	Major	Resolved
3	Duplicate input fields would be overwritten	Minor	Resolved
4	`DumpState` query does not paginate proposal modules which can result in out of gas errors	Minor	Acknowledged
5	Admin can replay migration to overwrite `dao_uri` and re-enable proposal modules when contract is paused	Minor	Resolved
6	DAO has influence over voting results	Minor	Acknowledged
7	Too many proposal modules stored would cause migration to fail	Minor	Acknowledged
8	Setting `AbsoluteCount` to zero will always show proposals as active	Minor	Resolved
9	A counterparty in escrow can always withdraw, resulting in temporary grief	Minor	Acknowledged
10	Active threshold could be set for a very small percentage	Minor	Acknowledged
11	All implemented electoral schemes are condorcet-incomplete	Minor	Resolved
12	Migration does not update the latest contract name and version	Minor	Resolved
13	Users can unstake zero amounts of funds	Minor	Resolved
14	Permissionless proposal creation policy could result in spam proposals	Minor	Acknowledged
15	`dao-voting-staking-denom-staked` allows malicious actors to manipulate voting power	Minor	Acknowledged
16	Non-token might be misconfigured as a staking asset	Informational	Resolved

17	Ranked choice voting not implemented	Informational	Resolved
18	Multiple choice voting is prone to social engineering attacks	Informational	Acknowledged
19	Iterations over hooks might run out of gas	Informational	Acknowledged
20	Proposal rationale can be modified after expiration	Informational	Acknowledged
21	Updated unstaking duration is not included in event	Informational	Resolved
22	Verifying negative decimal values is unnecessary	Informational	Resolved
23	Typographic errors and outstanding TODOs found	Informational	Resolved
24	Contracts should implement a two-step ownership transfer	Informational	Resolved
25	Condorcet proposal allows proposer to specify a large number of choices	Informational	Acknowledged
26	Duplicate messages are allowed to be proposed	Informational	Acknowledged

Detailed Findings

1. Queried voting power when executing proposals is set to current height, allowing anyone to execute proposals

Severity: Major

In `contracts/proposal/dao-proposal-multiple/src/contract.rs:365`, the caller's voting power is queried at the current block height to ensure they are one of the DAO's members. This implies that any caller can execute a proposal by staking a small number of funds after the proposal ends and calling the `execute_execute` function.

Recommendation

We recommend modifying line `365` to query the voting power at the proposal's start height to ensure only DAO members that participated in a poll can execute passed proposals.

Status: Resolved

The issue is resolved in `4ef0e216c52c7353387d1cdd6ccc966329fc14aa`.

2. Users might be unable to claim NFTs if they unstake a large amount

Severity: Major

In `contracts/voting/dao-voting-cw721-staked/src/contract.rs:143-158`, the `execute_unstake` function attempts to validate the outstanding NFT claims is not greater than or equal to the configured max claim amount. Since the outstanding claims do not include the current amount of `token_ids` to claim, a user can unstake a large number of NFTs in one transaction as long the old outstanding claim amount is smaller than the max claim limit. As a result, users would be unable to claim NFTs using the `execute_claim_nfts` function due to an out-of-gas error.

Please see the `test_bypass_max_claims` test case to reproduce the issue.

We classify this issue as major since it can only happen when the user unstakes a large number of NFTs in one transaction, but the user is not able to recover these NFTs.

Recommendation

We recommend validating that the number of outstanding claims and the length of `token_ids` do not exceed the configured max claim amount.

Status: Resolved

The issue is resolved in [362a8e402a08b5d241262b3d07db0bff6eb30c2d](362a8e402a08b5d241262b3d07db0bff6eb30c2d).

3. Duplicate input fields would be overwritten

Severity: Minor

In `contracts/dao-core/src/contract.rs:76-78`, the `InitialItem` key from the `msg.initial_items` vector is being saved into the `ITEMS` storage state. If there were duplicate keys, the previous ones would get overwritten, causing only the last value to be stored.

Besides that, the `execute_update_sub_daos_list` function in `contracts/dao-core/src/contract.rs:497-500` does not filter duplicate addresses in the `to_add` vector. Consequently, the previous `SubDao` charter would get overwritten, causing the `SUBDAO_LIST` only to save the last charter value.

Recommendation

We recommend deduplicating the `msg.initial_items` and `to_add` vector before storing the values.

Status: Resolved

The issue is resolved in [5b6f080c8476e079a8d5346081d69f57d9326c3d](5b6f080c8476e079a8d5346081d69f57d9326c3d).

4. `DumpState` query does not paginate proposal modules which can result in out of gas errors

Severity: Minor

In `contracts/dao-core/src/contract.rs:665-668`, the `query_dump_state` function attempts to fetch all proposal modules stored in the `PROPOSAL_MODULES` storage. The query will fail if too many stored proposal modules exist due to an out-of-gas error. As a result, the `active_proposal_module_count` and `total_proposal_module_count` values cannot be queried, representing the number of active proposal modules and the total number of proposal modules.

We classify this issue as minor since only the contract admin can instantiate proposal modules.

Recommendation

We recommend paginating the proposal modules during the `DumpState` query.

Status: Acknowledged

The client states that they acknowledge that the `DumpState` query may run out of gas. They use them purely as a way to get a lot of info in one query, which is otherwise accessible via sub-functions.

The front end will then try and dump the `DumpState` query and then fall back to the other queries, which are paginated if the `DumpState` fails. This "fixes" the issue in that it makes those fields accessible via a query that does not run out of gas.

5. Admin can replay migration to overwrite `dao_uri` and re-enable proposal modules when contract is paused

Severity: Minor

The `migrate` handler in `contracts/dao-core/src/contract.rs:831-879` does not ensure the current contract version is lower than the version to migrate to. This is problematic because the contract admin can abuse the migration function to modify the `dao_uri` and re-enable all proposal modules when the contract is paused, allowing them to partially bypass the contract state. Ideally, a paused contract should not have its storage state mutable until the expiration ends.

Please see the `test_replay_migrate` test case to reproduce the issue.

We classify this issue as minor since only the contract admin can perform migrations.

Recommendation

We recommend adding a check to ensure the current contract version is lower than the version to migrate.

Status: Resolved

The issue is resolved in `b46a49485b9da27ddfbf2c3a9efda2c1696a1b06`.

6. DAO has influence over voting results

Severity: Minor

In `contracts/proposal/dao-proposal-multiple/src/contract.rs:285`, the voter’s voting power is determined by querying the configured DAO contract. Since the total voting power is queried and stored as seen in line `207`, updating the DAO contract to a different address might cause incorrect voting results.

Suppose the voter staked more tokens in the newly updated DAO contract. This would cause the `execute_vote` function to query the new voting power and compare it against the old total voting power, leading to an incorrect voting result.

Besides that, the DAO can also update the `only_members_execute` boolean which affects the `execute_execute` function. Suppose the `only_members_execute` configuration is set to `true` during the proposal creation time and set to `false` after the proposal passes. The DAO members would expect only they can execute the proposal. However, since the `only_members_execute` value is loaded from the configuration directly, anyone can execute the proposal.

Lastly, the `close_proposal_on_execution_failure` boolean can be updated by the DAO to close proposals that fail to execute automatically. Suppose the `close_proposal_on_execution_failure` was set to `false` and the message to execute fails in line `409`. The DAO can interfere by updating the configuration to `true` and re-execute the function, causing the proposal status to be set to `ExecutionFailed`.

This issue is also present in the `contracts/proposal/dao-proposal-single` contract.

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

Recommendation

We recommend storing the DAO address, `only_members_execute`, and `close_proposal_on_execution_failure` configuration in the proposal struct to prevent the DAO from influencing voting results.

Status: Acknowledged

The client states that `only_members_execute` and `close_proposal_on_execution` failure are intentionally left under the control of the DAO as neither could DOS a DAO, and they are useful to change retroactively. This unit test demonstrates an example of where applying `close_proposal_on_execution_failure` retroactively is useful.

`only_members_execute` has the same property in that a DAO may want to enable it retroactively for proposals if they become concerned with the prospect of non-members executing past proposals. Generally speaking, it is the case that the DAO controls all aspects of proposals as it is the admin of its proposal contract. The client considers this a tradeoff worth making to allow the DAO to upgrade its modules.

7. Too many proposal modules stored would cause migration to fail

Severity: Minor

In `contracts/dao-core/src/contract.rs:837-859` and `contracts/proposal/dao-proposal-single/src/contract.rs:879-919`, all stored proposal modules are collected without pagination. This is problematic because if too many proposal modules were stored, the migrate function would fail due to an out-of-gas error, preventing the admin from migrating the contract to the latest version.

We classify this issue as minor since only the contract admin can instantiate proposal modules.

Recommendation

We recommend changing the migration logic into a state machine to guarantee the consistency of migrations. This could be achieved by automatically setting a `migration_pending` boolean when the migration process starts and automatically removing the `migration_pending` boolean once there are no proposal modules left to migrate.

Status: Acknowledged

The client states that they have measured gas usage during migrations, and as there are currently no DAOs deployed with more than one proposal module, they feel they are squarely within gas tolerances, and the additional complexity of a migration state machine has limited practical advantages.

8. Setting `AbsoluteCount` to zero will always show proposals as active

Severity: Minor

In `contracts/voting/dao-voting-cw20-staked/src/contract.rs:307-315`, no validation ensures the `count` value inside the `ActiveThreshold::AbsoluteCount` enum is greater than zero. In comparison, the `ActiveThreshold::Percentage` enum is validated to ensure the percent value is greater than zero in line `213`. If the `count` value is configured to zero, the `query_is_active` query will always return `true` despite the total amount of tokens staked being zero.

We classify this issue as minor since only the DAO contract can configure active thresholds.

Recommendation

We recommend adding a check in `assert_valid_absolute_count_threshold` to ensure the `count` value is greater than zero.

Status: Resolved

The issue is resolved in 8e12c3af3647ee18fbe321ef81beb8c56adffa11.

9. A counterparty in escrow can always withdraw, resulting in temporary grief

Severity: Minor

In `contracts/external/cw-token-swap/src/contract.rs:197`, the `execute_withdraw` function allows a counterparty to withdraw the funds deposited in the contract. Since the escrow completion depends on the payment of the last counterparty, whoever deposited first can always withdraw their funds from the contract (such as frontrunning the other user), causing the transaction never to be completed.

Recommendation

We recommend introducing a mechanism such as a minimum withdrawal period to prevent this attack.

Status: Acknowledged

The client states that they consider this temporary grief a feature of the contract. If a counterparty pulls out of their commitment immediately before the swap would clear, this is likely socially meaningful, and they think that should play out at the social layer.

10. Active threshold could be set for a very small percentage

Severity: Minor

In `contracts/voting/dao-voting-cw20-staked/src/contract.rs:214-229`, the `execute_update_active_threshold` function allows a DAO to set an active threshold. It validates that the threshold is between 0 to 100, meaning the DAO can set it to a very small percentage (e.g. 2%). This may allow malicious participants to execute an attack on the DAO.

Recommendation

We recommend requiring a minimum percentage that prevents abuse, for example 50%.

Status: Acknowledged

The client states they expect this threshold to take on very low values. Practically speaking, most DAOs allocate upwards of 90% of their treasury to the DAO, so the percentage staked in terms of the absolute number of tokens is expected to be a value south of 5% frequently. They feel this is a threshold best restricted in the frontend.

11. All implemented electoral schemes are Condorcet-incomplete

Severity: Minor

Both intended electoral systems, ranked choice voting and single-member plurality voting, do not satisfy the Condorcet winner criterion. This means both cannot ensure that the Condorcet winner (the alternative that would win a two-option vote against each of the other options in a plurality vote) wins the vote. A mechanism to pick a Condorcet winner is desirable because it is considered prudent decision-making, particularly regarding budgetary decisions.

Recommendation

We recommend implementing a voting mechanism that satisfies the Condorcet winner criterion. Note that if issue 10 is fixed by requiring a minimum percentage of at least 50%, this issue will be fixed automatically.

This is because the first-past-the-post voting combined with a 50% threshold is Condorcet complete. If one option receives more than 50% of all votes, it will win a 1:1 race against any other option. We flagged this issue because both the 50% threshold and the Condorcet completeness are design choices that might depend heavily on your users' preferences. It might well be that either a lower threshold combined with another voting scheme or simply a 50% threshold is sufficient for your users. Not picking the Condorcet winner is not per se bad, as there is theoretically no perfect (or “fair”) voting system, as the Arrow theorem dictates. It is just not the best practice to offer a range of systems with similar drawbacks.

Status: Resolved

The client implemented a Condorcet proposal contract, which has been included in this audit.

12. Migration does not update the latest contract name and version

Severity: Minor

In `contracts/voting/dao-voting-staking-denom-staked/src/contract.rs:116-118`, the migrate handler does not automatically set the latest contract name and version. If the admin migrates the contract, the `query_info` query in line `100` will still show the old contract name and version.

Recommendation

We recommend calling `set_contract_version` during the migrate function.

Status: Resolved

The issue is resolved in `0ef1a2ad3a932926bdda3372f626fe0864c1e01c` by removing the contract.

13. Users can unstake zero amounts of funds

Severity: Minor

In `contracts/voting/dao-voting-native-staked/src/contract.rs:180-197`, the `execute_unstake` function does not validate the amount to be 0. Since the amount is not validated, a valid claim is still created for the user, causing the outstanding claims to be increased in line `182`. Consequently, the current outstanding claims will be growing and compared to the `MAX_CLAIMS` amount, potentially causing the user to be unable to unstake tokens as intended.

Recommendation

We recommend adding a check to ensure the amount to unstake is greater than zero.

Status: Resolved

The issue is resolved in `1f599468b10e743814ccc0796eb9e3bd0d321fe0`.

14. Permissionless proposal creation policy could result in spam proposals

Severity: Minor

In `contracts/proposal/dao-proposal-single/src/contract.rs:164-166`, the DAO contract allows different configurations. If the `pre-propose` module is unset and the `proposal_creation_policy` is set to `Anyone`, malicious users can spam the DAO with unnecessary proposals.

Recommendation

We recommend ensuring that the `deposit_info` is not optional in the `pre-propose-base` module when the creation policy is set for `Anyone`.

Status: Acknowledged

The client states that the permissionless proposal creation state is useful in the event that a pre-propose module fails. In that event, the proposal module will evict the pre-propose module and allow anyone to propose such that the pre-propose error is recoverable.

15. `dao-voting-staking-denom-staked` allows malicious actors to manipulate voting power

Severity: Minor

In `contracts/voting/dao-voting-staking-denom-staked/src/contract.rs:58-98`, the `query_voting_power_at_height` and `query_total_power_at_height` function return the current voting power irrespective of the provided height argument.

Normally, the total voting power of a proposal is snapshotted at a specific block height using `TotalPowerAtHeight` in the voting contract. When a user votes, the `VotingPowerAtHeight` is queried with the proposal's start height to get the user's staked amount. Since both queries return the height based on the user's argument value, this means the user-staked amount can be manipulated.

If the user changed their staked amount (e.g., staked more tokens), their staking power would be changed. However, the total power saved in the proposal still uses the old total staking power value. Due to this, users who vote will be queried with current voting power, not snapshotted power. This allows malicious users to manipulate their voting weight.

For example, imagine Alice and Bob each have `500` voting power at block height `10`. The stored total voting power in the proposal will be `1000` (`500+500=1000`). At block height `20`, Alice stakes more tokens causing her voting power to be `1200`. Alice then votes, and the contract queries Alice's voting power at block height `10`. However, due to the issue described above, her voting power returned will be `1200` instead of `500`, causing Alice to have a higher voting power than intended.

We classify this issue as minor because ICS chains do not have a staking module, preventing the usage of this contract. However, this would be an issue if the contract is deployed on a chain with a staking module.

Recommendation

We recommend correctly determining users' voting power based on the proposal creation height.

Status: Acknowledged

The client may implement a limit to the number of hook receivers in the future.

16. Non-token might be misconfigured as a staking asset

Severity: Informational

In `contracts/staking/cw20-stake/src/contract.rs:79`, any address could be configured as a staking token, even contracts do not implement token interfaces.

We classify this issue as informational because the contract instantiator can always misconfigure an incorrect CW20 token address, even if the recommendation below is implemented.

Recommendation

We recommend validating that the `token_address` is configured as a staking asset by querying the `TokenInfo`.

Status: Resolved

The issue is resolved in `1c15e58f484ba9caf76e235733ecd9ad37bb630f`.

17. Ranked choice voting not implemented

Severity: Informational

The documentation mentions the implementation of ranked choice voting, but it is not implemented.

Recommendation

We recommend removing ranked choice voting from current documentation or explaining that it is forthcoming.

Status: Resolved

18. Multiple choice voting is prone to social engineering attacks

Severity: Informational

Multiple choice voting systems, where users can specify options as free text, are prone to social engineering attacks, as users can try to confuse users with odd alternatives. This is particularly problematic in conjunction with Condorcet-incomplete voting schemes, where minority alternatives can win. Users can try to submit duplicate "No"-options or introduce "quasi-No"-options and split up the "No"-voters into smaller subsets such that they do not get a share of votes to prevent any of the other options.

An oversimplified example is illustrated below.

Question: “What kind of horse should we buy?”

Answers:

- “Black”
- “Red”
- “We should discuss this again next year”
- “None of the above”

While both “We should discuss this again next year” and “None of the above” are the equivalent of “We shouldn't buy a horse now”, one of the options “Black” or “Red” might win even if the majority of voters do not want to buy a horse, because the “No”-votes are split.

Recommendation

We recommend educating users and developing a style guide for multiple-choice voting. Alternatively, a more complex voting scheme, such as the Schulze method, might resolve the issue. As there are no complete fixes to social engineering attacks, we classify this issue as informational.

Status: Acknowledged

The client states they expect such anti-social-engineering changes to be implemented in the frontend, and they’ll continue to explore ways of doing so. As they have no frontend for multiple choice voting yet, this will be done in the future.

19. Iterations over hooks might run out of gas

Severity: Informational

In `packages/cw-hooks/src/lib.rs:34-52`, iterations over `hooks` might run out of gas.

We classify this issue as informational as only admins can cause it and it is recoverable.

Recommendation

We recommend limiting the number of hooks.

Status: Acknowledged

The client states that hook receivers can consume arbitrary amounts of gas, and gas limits can change due to on-chain governance. They feel that picking a meaningful limit for the number of hook receivers is not possible.

20. Proposal rationale can be modified after expiration

Severity: Informational

In `contracts/proposal/dao-proposal-single/src/contract.rs:450-471`, no validation ensures the voter cannot update their rationale after the proposal ended. This prevents the contract from recording correct on-chain preserves of voters' rationale for ended proposals.

We classify this issue as informational since there is no security risk.

Recommendation

We recommend checking whether the proposal is expired when calling `execute_update_rationale`.

Status: Acknowledged

The client states that they store the record of proposal rationales on-chain so that a historical view of the rationales for a vote may be viewed in the frontend. They believe that changing one's mind is an important part of governance and want to allow for that in the contracts.

21. Updated unstaking duration is not included in event

Severity: Informational

In `contracts/voting/dao-voting-cw721-staked/src/contract.rs:208-231`, the contract owner can update the owner and duration value. However, the emitted event does only record the updated owner value, as seen in line `225`. The updated unstaking duration is not included in the event attributes, potentially causing off-chain event listeners to fail to index the latest unstaking duration set by the contract owner.

Recommendation

We recommend adding the unstaking duration to the event attributes.

Status: Resolved

The issue is resolved in [`e81708b4bf224ceca68cd4e005b440b07d665090`]().

22. Verifying negative decimal values is unnecessary

Severity: Informational

In `contracts/voting/dao-voting-cw20-staked/src/contract.rs:213`, the `percent` value is checked whether it is less than or equal to `Decimal::percent(0)`. Since the `Decimal` value used by `percent` is an unsigned integer, negative values will automatically underflow into a positive integer, so the former validation is unnecessary.

Recommendation

We recommend modifying the second comparison in line `213` into "`percent == Decimal::percent(0)`" to prevent checking negative values for unsigned decimal values.

Status: Resolved

The issue is resolved in [`18523ca6479108ec6898a65e859a144bd2e076c5`]().

23. Typographic errors and outstanding TODOs found

Severity: Informational

In several instances of the codebase, typographical errors and outstanding TODOs were found during the audit:

- `contracts/external/cw-token-swap/src/error.rs:28`
- `packages/dao-voting/src/multiple_choice.rs:93`
- `contracts/pre-propose/dao-pre-propose-approval-single/src/contract.rs:105-107`

This affects the readability of the contracts.

Recommendation

We recommend correcting the typographic errors above and resolving the TODO comments.

Status: Resolved

The issue is resolved in [`2aeea7464ce7fd484320269067a9149b01d7ee6d`]().

24. 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: Resolved

The issue is resolved in `1673241909dccc57a561f5be9c06e5e21757be09`.

25. Condorcet proposal allows proposer to specify a large number of choices

Severity: Informational

In `contracts/proposal/dao-proposal-condorcet/src/contract.rs:104`, there is no max limit of choices enforced when executing the `Propose` message. If a proposal contains too many choices to vote on, it facilitates social engineering attacks and strategic voting possibilities.

As described in issue 18, social engineering attacks might rely on confusing or duplicate alternatives. As the number of options increases, it is more complex to understand the relationship between a variety of options and make educated choices, which facilitates social engineering.

At the same time, with more options and heterogeneous preferences among the voters, it will be easier for a small group of strategic voters to collude and create a situation where the preferences are intransitive such that the Condorcet method does not deliver a Condorcet winner, if an undesired outcome would occur otherwise. For further reference, consider the Wikipedia article Independence of irrelevant alternatives.

We classify this issue as informational because strategic voting will always be possible.

Recommendation

We recommend implementing a max number of choices similar to the `dao-proposal-multiple` contract in `packages/dao-voting/src/multiple_choice.rs:129-133`. Alternatively, another option to reduce strategic voting behavior would be to allow for secret ballots. This would make it harder for the colluding group to determine if and how to submit strategic votes because they lack information about the preferences of the other voters.

Status: Acknowledged

26. Duplicate messages are allowed to be proposed

Severity: Informational

In `contracts/proposal/dao-proposal-condorcet/src/contract.rs:100` and `contracts/proposal/dao-proposal-multiple/src/contract.rs:151`, there is no validation that ensures the `CosmosMsg` inside the vector of choices is different. This implies that the voter's preferences might not matter if the messages to execute are the same.

Recommendation

We recommend validating that the provided messages for different choices are not the same.

Status: Acknowledged

Appendix A: Test Cases

1. Test case for "Admin can replay migration to overwrite `dao_uri` and re-enable proposal modules"

The test case should fail if the vulnerability is patched.

```rust
#[test]
fn test_replay_migrate() {
    // reproduced in contracts/dao-core/src/tests.rs
    use cw_core_v1 as v1;

    let mut app = App::default();
    let govmod_id = app.store_code(sudo_proposal_contract());
    let voting_id = app.store_code(cw20_balances_voting());
    let core_id = app.store_code(cw_core_contract());
    let v1_core_id = app.store_code(v1_cw_core_contract());
    let cw20_id = app.store_code(cw20_contract());

    let proposal_instantiate = dao_proposal_sudo::msg::InstantiateMsg {
        root: CREATOR_ADDR.to_string(),
    };
    let voting_instantiate = dao_voting_cw20_balance::msg::InstantiateMsg {
        token_info: dao_voting_cw20_balance::msg::TokenInfo::New {
            code_id: cw20_id,
            label: "DAO DAO voting".to_string(),
            name: "DAO DAO".to_string(),
            symbol: "DAO".to_string(),
            decimals: 6,
            initial_balances: vec![cw20::Cw20Coin {
                address: CREATOR_ADDR.to_string(),
                amount: Uint128::from(2u64),
            }],
            marketing: None,
        },
    };

    const OAK: &str = "oak";

    // let oak become the contract admin
    let v1_core_instantiate = v1::msg::InstantiateMsg {
        admin: Some(OAK.to_string()),
        name: "DAO DAO".to_string(),
        description: "A DAO that builds DAOs.".to_string(),
        image_url: None,
        automatically_add_cw20s: false,
        automatically_add_cw721s: false,
        voting_module_instantiate_info: v1::msg::ModuleInstantiateInfo {
            code_id: voting_id,
            msg: to_binary(&voting_instantiate).unwrap(),
            admin: v1::msg::Admin::CoreContract {},
            label: "voting module".to_string(),
        },
        proposal_modules_instantiate_info: vec![
            v1::msg::ModuleInstantiateInfo {
                code_id: govmod_id,
                msg: to_binary(&proposal_instantiate).unwrap(),
                admin: v1::msg::Admin::CoreContract {},
                label: "governance module 1".to_string(),
            },
            v1::msg::ModuleInstantiateInfo {
                code_id: govmod_id,
                msg: to_binary(&proposal_instantiate).unwrap(),
                admin: v1::msg::Admin::CoreContract {},
                label: "governance module 2".to_string(),
            },
        ],
        initial_items: None,
    };

    // init contract with creator as migration admin
    let core_addr = app
        .instantiate_contract(
            v1_core_id,
            Addr::unchecked(CREATOR_ADDR),
            &v1_core_instantiate,
            &[],
            "cw-governance",
            Some(CREATOR_ADDR.to_string()),
        )
        .unwrap();

    // creator performs normal migration
    app.execute(
        Addr::unchecked(CREATOR_ADDR),
        CosmosMsg::Wasm(WasmMsg::Migrate {
            contract_addr: core_addr.to_string(),
            new_code_id: core_id,
            msg: to_binary(&MigrateMsg::FromV1 { dao_uri: None }).unwrap(),
        }),
    )
    .unwrap();

    // oak disables one proposal module
    app.execute_contract(
        Addr::unchecked(OAK),
        Addr::unchecked(core_addr.to_string()),
        &ExecuteMsg::ExecuteAdminMsgs {
            msgs: vec![WasmMsg::Execute {
                contract_addr: core_addr.to_string(),
                msg: to_binary(&ExecuteMsg::UpdateProposalModules {
                    to_add: vec![],
                    to_disable: vec!["contract3".to_string()] })
                    .unwrap(),
                funds: vec![],
            }
            .into()],
        },
        &[],
    ).unwrap();

    // oak pauses the contract
    app.execute_contract(
        Addr::unchecked(OAK),
        Addr::unchecked(core_addr.to_string()),
        &ExecuteMsg::ExecuteAdminMsgs {
            msgs: vec![WasmMsg::Execute {
                contract_addr: core_addr.to_string(),
                msg: to_binary(&ExecuteMsg::Pause {
                    duration: Duration::Height(10),
                })
                .unwrap(),
                funds: vec![],
            }
            .into()],
        },
        &[],
    ).unwrap();

    // cannot do anything when paused
    app.execute_contract(
        Addr::unchecked(OAK),
        Addr::unchecked(core_addr.to_string()),
        &ExecuteMsg::ExecuteAdminMsgs {
            msgs: vec![WasmMsg::Execute {
                contract_addr: core_addr.to_string(),
                msg: to_binary(&ExecuteMsg::UpdateConfig {
                    config: Config {
                        dao_uri: None,
                        name: "The Empire Strikes Back".to_string(),
                        description: "haha lol we have pwned your
DAO".to_string(),
                        image_url: None,
                        automatically_add_cw20s: true,
                        automatically_add_cw721s: true,
                    },
                })
                    .unwrap(),
                    funds: vec![],
                }
                .into()],
            },
            &[],
        ).unwrap_err();

    // creator replay migrate function to update dao_uri and enable all proposal
modules
    let bad = Some("malicious".to_string());
    app.execute(
        Addr::unchecked(CREATOR_ADDR),
        CosmosMsg::Wasm(WasmMsg::Migrate {
            contract_addr: core_addr.to_string(),
            new_code_id: core_id,
            msg: to_binary(&MigrateMsg::FromV1 { dao_uri: bad.clone()
}).unwrap(),
        }),
    )
    .unwrap();

    // dao_uri is modified in PAUSED state
    let dao_uri: DaoURIResponse = app
        .wrap()
        .query_wasm_smart(core_addr.clone(), &QueryMsg::DaoURI {})
        .unwrap();
    assert_eq!(dao_uri.dao_uri, bad);

    // proposal module can be re-enabled in PAUSED state
    let new_state: DumpStateResponse = app
        .wrap()
        .query_wasm_smart(core_addr.clone(), &QueryMsg::DumpState {})
        .unwrap();

    let proposal_modules = new_state.proposal_modules;
    for (idx, module) in proposal_modules.iter().enumerate() {
        let prefix = derive_proposal_module_prefix(idx).unwrap();
        assert_eq!(prefix, module.prefix);
        assert_eq!(ProposalModuleStatus::Enabled, module.status);
    }
}
```

2. Test case for “Users might be unable to claim NFTs if they unstake a large amount”

The test case should fail if the vulnerability is patched.

```rust
#[test]
fn test_bypass_max_claims() -> anyhow::Result<()> {
    // reproduced in
contracts/voting/dao-voting-cw721-staked/src/testing/tests.rs
    let CommonTest {
        mut app,
        module,
        nft,
    } = setup_test(None, Some(Duration::Height(1)));

    let mut to_stake = vec![];

    for i in 1..201 {
        let i_str = &i.to_string();
        mint_and_stake_nft(&mut app, &nft, &module, CREATOR_ADDR, i_str)?;

        if i <= 99 { // unstake first 99 NFTs
            unstake_nfts(&mut app, &module, CREATOR_ADDR, &[i_str])?;
        } else { // push rest of NFT ids to vec
            to_stake.push(i_str.clone());
        }
    }

    let binding = to_stake.iter().map(|s| s.as_str()).collect::<Vec<_>>();
    let to_stake_slice: &[&str] = binding.as_slice();

    // unstake in one tx to bypass MAX_CLAIMS limit
    unstake_nfts(&mut app, &module, CREATOR_ADDR, to_stake_slice)?;

    let claims = query_claims(&app, &module, CREATOR_ADDR)?;
    assert_eq!(claims.nft_claims.len(), 200);

    Ok(())
}
```