Zellic

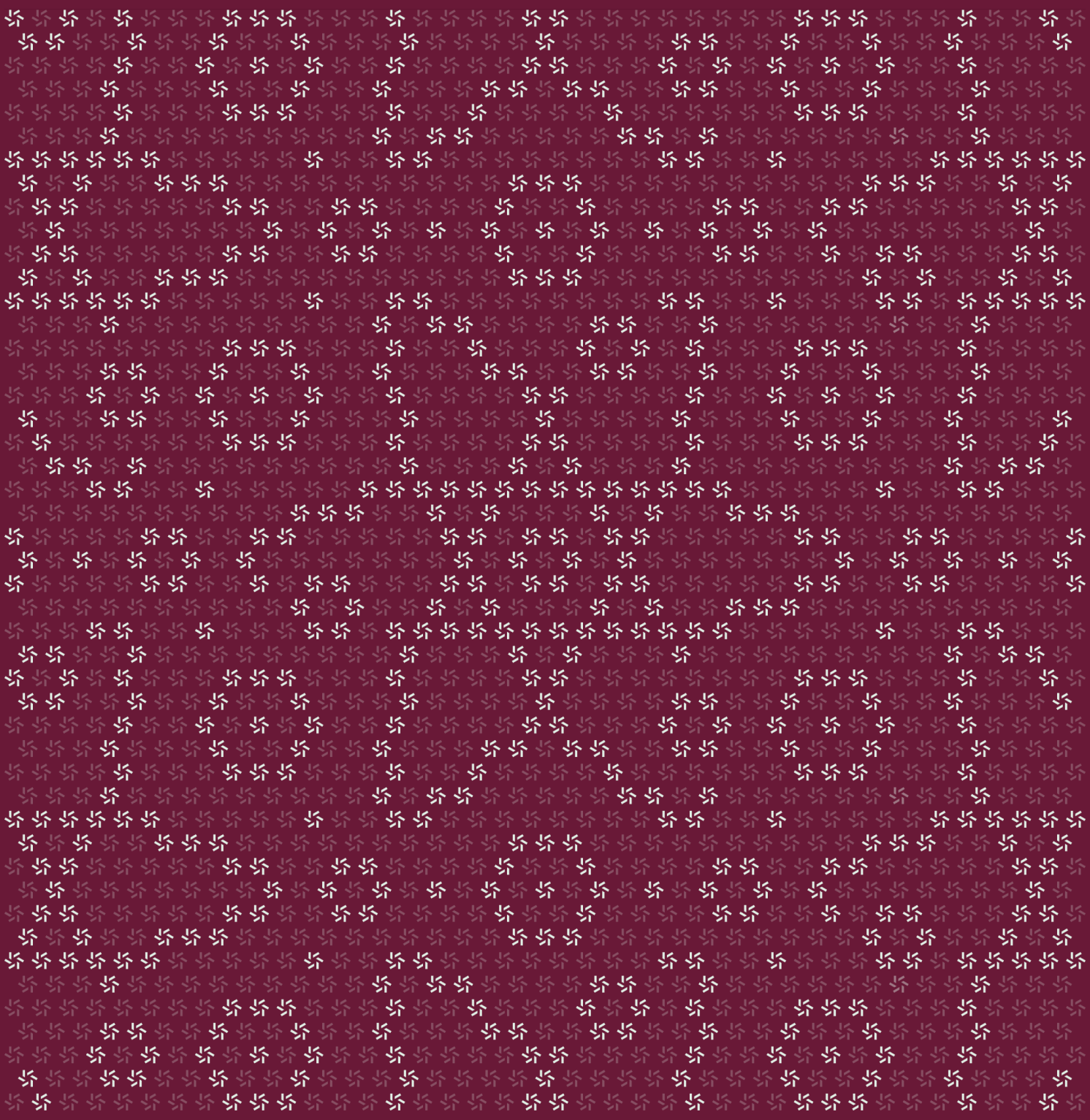
Prepared for
Noah Saso
DAO DAO

Prepared by
Varun Verma
Nan Wang
Zellic

February 19, 2025

DAO DAO

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for DAO DAO from February 6th to February 12th, 2025. During this engagement, Zellic reviewed DAO DAO's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could an adversary exploit the delegation logic or historical storage mechanisms to cause unexpected overflows, exceed gas limits, or otherwise disrupt normal DAO operations?
- Does the vote-delegation system accurately handle fractional delegations and overrides without introducing potential miscounting or unexpected behavior?
- Do the CosmWasm SnapshotMap and related mechanisms reliably track historical state?
- Are there any hidden risks in how this delegation system directly modifies or integrates with the proposal modules, potentially affecting the DAO's most critical voting processes?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped DAO DAO contracts, we discovered two findings. No critical issues were found. One finding was of high impact and the other finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of DAO

DAO in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	1
Medium	0
Low	0
Informational	1

2. Introduction

2.1. About DAO DAO

DAO DAO contributed the following description of DAO DAO:

> DAO DAO is a suite of open-source interchain DAO tooling, enabling projects, protocols, and communities to build composable and transparent governance systems without any code.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational"

finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

DAO DAO Contracts

Type	Rust
Platform	Cosmos
Target	dao-contracts
Repository	https://github.com/DA0-DA0/dao-contracts ↗
Version	`4913762249365a4eb9489d67d705f7bc188e996f`

Programs	`contracts/delegation/dao-vote-delegation/**`
	`packages/cw-snapshot-vector-map/**`
	`packages/dao-voting/src/delegation.rs`
	`contracts/proposal/dao-proposal-single/src/contract.rs`
	`contracts/proposal/dao-proposal-multiple/src/contract.rs`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.5 person-weeks. The assessment was conducted by two consultants over the course of one calendar week.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Varun Verma
Engineer
varun@zellic.io

Nan Wang
Engineer
nan@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 6, 2025	Kick-off call
February 6, 2025	Start of primary review period
February 12, 2025	End of primary review period

3. Detailed Findings

3.1. Mismatched tally updates causing underflow in revoting

Target	dao-proposal-single/src/contract.rs		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

There are two separate tallies for user votes:

1. **`prop.votes`** — tracks total voting power, including delegations
2. **`prop.individual_votes`** — tracks only the user's direct (nondelegated) power

When removing a revote, the contract subtracts `current_ballot.power` (including delegations) from both tallies, even though `prop.individual_votes` is incremented only by the user's direct voting power. This discrepancy risks causing an arithmetic underflow in `prop.individual_votes`.

Here are the potentially problematic lines:

```
// Remove side
prop.individual_votes.remove_vote(current_ballot.vote, current_ballot.power);

// Add side
prop.individual_votes.add_vote(vote, vote_power.individual);
```

Impact

This may lead to the following two issues.

1. **Underflow risk.** Subtracting a larger delegated total from a smaller direct-power tally can create arithmetic underflows.
2. **Data integrity.** Mismatched additions and removals may produce incorrect tallies, affecting proposal outcomes.

Recommendations

Ensure `individual_votes` increments and decrements by the same (individual) power amount.

Remediation

This issue has been acknowledged by DAO DAO, and a fix was implemented in PR #900.

3.2. SnapshotVectorMap does not enforce block-height ordering

Target	packages/cw-snapshot-vector-map/src/lib.rs		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

SnapshotVectorMap relies on data being written in increasing block height order to create snapshots. However, the current `push` function does not enforce this ordering, and there is no runtime check to ensure that the `curr_height` passed in is strictly increasing. This could lead to data being recorded out of chronological order, resulting in potentially inaccurate historical snapshots if developers inadvertently insert data at lower block heights after higher ones.

```
/// Adds an item to the vector at the current block height, optionally
/// expiring in the future, returning the ID and potentially the expiration
/// height of the new item, along with the total number of items. This block
/// should be greater than or equal to the blocks all previous items were
/// added/removed at. Pushing to the past will lead to incorrect behavior.
pub fn push(
    &self,
    store: &mut dyn Storage,
    k: &K,
    data: &V,
    curr_height: u64,
    expire_in: Option<u64>,
) -> StdResult<((u64, Option<u64>), usize)> {
    // get next ID for the key, defaulting to 0
    let next_id = self
        .next_ids
        .may_load(store, k.clone())?
        .unwrap_or_default();

    // add item to the list of all items for the key
    self.items.save(store, &(k.clone(), next_id), data)?;

    // get active list for the key
    let mut active = self.active.may_load(store,
    k.clone())?.unwrap_or_default();

    // remove expired items
    active.retain(|(_, expiration)| {
```

```
            expiration.map_or(true, |expiration| expiration > curr_height)
        });

        // add new item and save list
        let expiration = expire_in.map(|d| curr_height + d);
        active.push((next_id, expiration));

        // save the new list
        self.active.save(store, k.clone(), &active, curr_height)?;

        // update next ID
        self.next_ids.save(store, k.clone(), &(next_id + 1))?;

        Ok(((next_id, expiration), active.len()))
    }
```

Impact

Pushing to the past will lead to incorrect behavior, potentially corrupting historical snapshots.

Recommendations

Introduce a runtime check in the `push` function or its callers to verify that new entries are always appended at block heights greater than or equal to the last recorded block height, thereby preventing nonsequential data insertion.

Remediation

This issue has been acknowledged by DAO DAO, and a fix was implemented in PR #900 ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Erroneous comment in the `get_udvp` function

The comment in the `get_udvp` function regarding `UNVOTED_DELEGATED_VP` is inconsistent with the actual logic. The original comment suggests that whenever the delegate or any of their delegators cast a vote, `UNVOTED_DELEGATED_VP` will be updated. However, based on the code-execution path in `execute_vote_hook`, `UNVOTED_DELEGATED_VP` is only updated when a delegator votes.

To address this issue, DAO DAO has updated the relevant comment to the following, ensuring a more accurate reflection of the actual behavior:

```
// total delegated VP at that height. UNVOTED_DELEGATED_VP gets set when the
// delegate or one of their delegators casts a vote. if empty, none of them
// have voted yet.
// total delegated VP at that height. UNVOTED_DELEGATED_VP gets set when one
// of their delegators casts a vote. If empty, none of them have voted yet.
```

This revised comment aligns with the underlying code logic, helping to reduce confusion and improve the clarity and maintainability of the implementation.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Component: Delegation module

Description

The DAO Vote Delegation module provides functionality for members of a DAO to delegate their voting power to other registered delegates. They designate a portion (or all) of their voting power to a registered delegate. Once delegated, the delegate can vote on behalf of those delegators. It also ensures that delegates themselves cannot redelegate any voting power they hold and handles expiring delegations if configured.

The following represent the core actions in the system:

- **Registering and unregistering (`execute::Register` / `execute::Unregister`).** A user can mark themselves as a delegate if they hold nonzero voting power. They can also remove their registration status.
- **Delegating (`execute::Delegate`).** A delegator assigns a percentage of their voting power to a specific delegate. The code checks for basic constraints such as not exceeding 100% total delegations, having available voting power, and not allowing delegates to further delegate.
- **Undelegating (`execute::Undelegate`).** A delegator removes or reduces an existing delegation, instantly reducing the delegate's delegated voting power.
- **Accepting voting-power hooks (`execute::StakeChangeHook` / `execute::NftStakeChangeHook` / `execute::MemberChangedHook` / `execute::VoteHook`).** The contract is designed to accept hooks from staking contracts or other modules to keep delegated voting power consistent when a delegator's base voting power changes.

Invariants

- More than 100% cannot be delegated. A delegator's total delegations (sum of all percentages) must not exceed 100%.
- Delegates cannot redelegate. Once a user registers as a delegate, they cannot delegate their own voting power to someone else, ensuring no circular delegation loops.
- There is the max-delegations constraint. The delegator cannot exceed the configured

`max_delegations`.

- Valid voting power — a user must have nonzero voting power to register as a delegate.

Test coverage

Cases covered

- Basic delegation flows (register, delegate, undelegate)
- Vote-power caps and updates
- Delegation expiration and validity periods
- Vote overrides and proposal tallying
- Multiple-delegate registration/unregistration
- Unauthorized access attempts
- Contract migration
- Gas-limit scenarios with many delegates
- Edge cases around delegate-registration timing
- Member voting-power changes

Cases not covered

- Complex token staking/unstaking patterns

Attack surface

Only the DAO address can update the module configuration, as an attacker who tries to call `UpdateConfig` without being the DAO will fail. Only a registered hook caller can invoke stake or voting-power change hooks, and any other caller is rejected, mitigating this risk. The code also disallows a delegate from delegating further, preventing chain loops. The module receives cross-contract calls from the DAO but checks the caller's address to make sure it is known.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the CosmWasm-capable networks.

During our assessment on the scoped DAO DAO contracts, we discovered two findings. No critical issues were found. One finding was of high impact and the other finding was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.