Audit Report

Dora Vota

v1.0

June 13, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Matsushiba Factory Pte Ltd. to perform a security audit of the Dora Vota Cosmos SDK chain.

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/DoraFactory/doravota
Commit	`0dae08469da2bcb021d85ce2ab83d5abd50645c9`
Scope	All the code related to the baseapp was in scope.
Fixes verified at commit	`e1e8c7bd251a4cd828bd26f1b442b8e23338a1b7` 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

Dora Vota is a specialized blockchain designed for voting and decentralized community governance. It facilitates voting rounds for various communities and organizations, ensuring secure and transparent decision-making processes.

The audit scope is restricted to the baseapp.

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	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium	The README file and comments describe most of the architectural design choices.
Test coverage	Low	Neither unit tests nor integration tests have been implemented.

Summary of Findings

No	Description	Severity	Status
1	Infinite gas allocated per block allows attackers to halt the chain	Critical	Acknowledged
2	Inconsistency between `evidence` and `staking` parameters allows malicious validators to escape penalties	Major	Acknowledged
3	Usage of deprecated `x/crisis` module allows attackers to execute DoS attacks	Major	Acknowledged
4	Oversized maximum block size could allow DoS attacks	Major	Acknowledged
5	Vulnerability in `wasmvm` package	Major	Resolved
6	Vulnerabilities in `cosmos/cosmos-sdk` package	Minor	Resolved
7	Unhandled errors in the codebase	Informational	Resolved

Detailed Findings

1. Infinite gas allocated per block allows attackers to halt the chain

Severity: Critical

In the mainnet genesis file, the `max_gas` parameter for a block is set to `-1` in `config/mainnet/genesis.json:8`.

In the context of the Cosmos SDK `baseapp` it leads to the usage of an `infiniteGasMeter` during block handling, see this code reference for details.

Consequently, this configuration permits unlimited gas usage per block, a setting typically reserved for test environments due to its potential to disrupt network operations by allowing transactions to consume excessive resources impacting the stability and performance of the blockchain and in the worst case causing a chain halt.

Recommendation

We recommend setting a finite `max_gas` limit per block in the mainnet genesis file to ensure controlled resource usage and maintain network stability. This change should be tested thoroughly to find an optimal gas limit that balances transaction throughput and system performance.

Status: Acknowledged

The client states that the parameter will be updated on-chain through a governance proposal.

2. Inconsistency between `evidence` and `staking` parameters allows malicious validators to escape penalties

Severity: Major

In the `staking` module configuration of the mainnet genesis file, the `unbonding_time` is set to `1,814,400` seconds (21 days).

However, in `config/mainnet/genesis.json:12` the `max_age_duration` within the `evidence` module parameters is set to `172,800,000,000,000` nanoseconds (2 days), which is a magnitude shorter.

Additionally, in `config/mainnet/genesis.json:11`, the `max_age_num_blocks` is configured to `100,000`, which translates to `600,000` seconds (approximately `6.94` days), assuming a 6-second block time. This duration also falls short of the specified `unbonding_time`.

These discrepancies mean evidence can only be reported during a fraction of the unbonding period, allowing validators who have committed offenses to escape penalties if the evidence expires before the end of the unbonding period.

Recommendation

We recommend adjusting the `max_age_duration` and `max_age_num_blocks` in the `evidence` parameters to match at least the `unbonding_time` defined in the `staking` parameters.

Status: Acknowledged

The client states that the parameter will be updated on-chain through a governance proposal.

3. Usage of deprecated `x/crisis` module allows attackers to execute DoS attacks

Severity: Major

The appchain currently employs the `x/crisis` module to allow any participant to halt the chain in the event of an invariant violation by sending a `MsgVerifyInvariant`. This mechanism is intended to increase the robustness of the network by enabling the detection of broken invariants.

However, the module is deprecated as indicated in GHSA-qfc5-6r3j-jj22 and GHSA-w5w5-2882-47pc because it fails to induce a panic within transaction processing, thus treating broken invariants as reverted transactions.

Processing `MsgVerifyInvariant` messages incurs significant computational overhead; however, the fee does not align with the computational demand, making these transactions cheaper relative to their processing cost.

This can be exploited by attackers to perform a Denial of Service (DoS) attack by flooding the network with these messages.

Synthetic testing revealed up to a 20% increase in CPU usage on nodes flooded with such messages.

Recommendation

We recommend removing the `x/crisis` module from the appchain configuration. Instead, simulation tests should be enhanced and the implementation of the `x/circuit` module should be considered.

Status: Acknowledged

The client states that the module will be removed in a future version.

4. Oversized maximum block size could allow DoS attacks

Severity: Major

In `config/mainnet/genesis.json:7`, the mainnet genesis configuration defines the `max_bytes` parameter for a block as `22,020,096` bytes (approximately 22 MB).

This significantly exceeds the typical sizes used by similar networks. For example, Osmosis uses a `max_bytes` parameter of 5 MB.

The large block size, as highlighted in GHSA-hq58-p9mv-338c, could potentially cause performance issues and make the network susceptible to Denial of Service (DoS) attacks due to increased processing and propagation times.

Recommendation

We recommend reducing the `max_bytes` setting for blocks to align with more typical configurations that balance throughput and network stability. This adjustment should consider the specific needs of the chain, such as transaction volume and block time targets.

Status: Acknowledged

The client states that no modifications will be implemented at this time, as it is necessary to maintain support for MACI smart contracts.

5. Vulnerability in `wasmvm` package

Severity: Major

The `wasmvm` package version used in the appchain is vulnerable to CWA-2023-004 since it allows permissionless wasm code upload.

This vulnerability allows untrusted Wasm submissions. Once exploited, it disrupts the confirmation of new transactions on-chain, leading to a potential network-wide outage.

Recommendation

We recommend updating the `wasmvm` package to version `v1.5.1`.

Status: Resolved

6. Vulnerabilities in `cosmos/cosmos-sdk` package

Severity: Minor

The `cosmos/cosmos-sdk` package version used in the appchain is vulnerable to slashing evasion, as reported in GHSA-86h5-xcpx-cfqc.

"An issue was identified in the slashing mechanism that may allow for the evasion of slashing penalties during a slashing event. If a delegation contributed to byzantine behavior of a validator, and the validator has not yet been slashed, it may be possible for that delegation to evade a pending slashing penalty through re-delegation behavior. Additional validation logic was added to restrict this behavior."

Similarly, it is vulnerable to a reduction of block production as reported in GHSA-2557-x9mg-76w8.

"When using the default PrepareProposalHandler and the default SenderNonceMempool, an issue was identified which may allow invalid blocks to be proposed when a single sender includes multiple transactions with non-sequential sequence numbers in certain conditions. If this state is reached, it can lead to a reduction in block production for a network."

Consequently, attackers can leverage these vulnerabilities to perform attacks against the network.

Recommendation

We recommend updating the `cosmos/cosmos-sdk` package to version `v0.47.10`.

Status: Resolved

7. Unhandled errors in the codebase

Severity: Informational

In `docs/docs.go:29` and `cmd/dorad/cmd/root.go:230`, the errors from the `ParseFS` and `overwriteFlagDefaults` functions are ignored when registering the OpenAPI service and overwriting default flag values.

Consequently, there will be silent failures for the above errors, causing issues when accessing the OpenAPI service and performing transactions from the command line.

Recommendation

We recommend panicking when an error occurs.

Status: Resolved