Security Audit Report

Dora Vota Sponsor Module

v1.0

November 13, 2025

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 GmbH has been engaged by Matsushiba Factory Pte Ltd. to perform a security audit of Dora Vota Sponsor Module.

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	`e3fcf8ff97e362ec48709234ccb7b23ac09f69e0`
Scope	The scope is restricted to the `x/sponsor-contract-tx` module.
Fixes verified at commit	`e150d7887efc6f8f561609144ea80e42c7fc4368` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The `x/sponsor-contract-tx` module provides contract-scoped fee sponsorship for CosmWasm `MsgExecuteContract` transactions.

It maintains a registry that maps a contract to a deterministically derived sponsor address plus per-user grant limits, tracks per-user usage, and exposes admin-only management messages to set, update, delete sponsorships, withdraw sponsor funds, and update parameters.

Integrated into the ante handler, it validates sponsored transactions (single signer; all messages target the same contract), optionally queries the contract's `check_policy` for eligibility under a gas cap, gives precedence to native feegrant, requires the user to have insufficient spendable balance, enforces per-user grant limits, and, if eligible, deducts fees from the sponsor address and updates usage; otherwise it falls back to standard fee processing.

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**, **Partially Resolved**, 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-High**	The client shared detailed documentation about the module in scope.
Test coverage	**Medium**	Test coverage computed by `go test` is `60.50%`

Summary of Findings

No	Description	Severity	Status
1	Non-deterministic map iteration in message extraction can cause consensus divergence	Critical	Resolved
2	Spam and resource-exhaustion risks	Major	Resolved
3	Sponsorship system does not prevent abuse	Minor	Resolved
4	Contract admin changes will result in loss of control of sponsorship funds	Minor	Resolved
5	Multiple `MsgExecuteContract` allowed without a cap in sponsored transactions	Minor	Resolved
6	Incomplete genesis state validation leading to potential state inconsistencies	Minor	Resolved
7	Unsafe error formatting with raw user addresses enables log amplification and log injection	Minor	Resolved
8	Sponsorship creation enables abuse	Minor	Resolved
9	Unsafe marshalling of policy query payloads	Minor	Resolved
10	Redundant user balance check in sponsored fee payment section	Informational	Resolved
11	Query all-sponsors lacks pagination support	Informational	Resolved
12	`SponsorshipEnabled` check should occur at the beginning of the antehandle to avoid unnecessary computation	Informational	Resolved
13	The `getUserAddressForSponsorship` is called multiple times for the same transaction causing redundant computation	Informational	Resolved
14	Replace hardcoded denom name	Informational	Resolved
15	Unused error `ErrInvalidPolicyResponse`	Informational	Resolved

Detailed Findings

1. Non-deterministic map iteration in message extraction can cause consensus divergence

Severity: Critical

In `x/sponsor-contract-tx/keeper/keeper.go:186-217`, the `extractAllContractMessages` parses `execMsg.Msg` into a `map[string]interface{}` and then selects the "first" top-level key via a for range over the map.

Go's map iteration order is non-deterministic across processes and runs. Different validators may therefore pick different keys (or produce differently ordered JSON when re-marshalling), yielding divergent `ContractMessage{MsgType, MsgData}` values.

Consequently, this non-determinism can cascade into state divergence across multiple nodes and a chain halt.

Recommendation

We recommend avoiding map iterations and parsing the data into a deterministic structure.

Status: Resolved

2. Spam and resource-exhaustion risks

Severity: Major

The ante handler in `x/sponsor-contract-tx` performs potentially expensive operations, including smart-contract policy queries, which can be abused to mount denial-of-service attacks against validators and the chain.

Under current sponsorship semantics, a failing transaction may result in no fees charged or funds deducted from the sender (for example, if the sender has no balance). Although each individual policy check is bounded by `MaxGasPerSponsorship`, an attacker can still submit large volumes of transactions using Sybil identities to repeatedly trigger the same costly queries at effectively zero economic cost.

The attacker may also use low-cost or intentionally failing transactions to occupy mempool and block space, forcing validators to perform many expensive checks and/or filling blocks with useless transactions.

This behavior wastes resources during `CheckTx`, and during `DeliverTx` if such transactions are included in blocks, increasing the probability of missed propose/sign rounds and other consensus delays.

Finally, attackers can exhaust sponsored budgets by generating many sponsored interactions (Sybil spam), denying legitimate sponsored users service.

Together, these vectors enable DoS modes that amplify validator load, deplete sponsorship funds, and exhaust block space with cheap or failing transactions.

Recommendation

We recommend ensuring that the ante handler expensive work has a real economic or accounting consequence and that sponsorship cannot be trivially drained or used to mount DoS.

Status: Resolved

3. Sponsorship system does not prevent abuse

Severity: Minor

In the `handleSponsorshipFallback` function in `x/sponsor-contract-tx/ante/ante.go:614-627`, when a user has insufficient balance and sponsorship is denied, the system returns an error but doesn't track these failed attempts.

This creates a potential abuse vector where attackers can repeatedly submit transactions with zero balance to trigger sponsorship checks, consume system resources, and potentially spam the network with failed transactions. Without tracking failed sponsorship attempts, there's no mechanism to detect or prevent users from repeatedly attempting to abuse the sponsorship system.

Recommendation

We recommend implementing a global tracking mechanism that records failed sponsorship attempts per user address and enforces a maximum number of attempts (e.g., 10) within a time window. Users who exceed this limit should be temporarily blocked from attempting sponsored transactions, with the counter resetting after a cooldown period. This would help prevent abuse while still allowing legitimate users who may temporarily have insufficient funds.

Status: Resolved

4. Contract admin changes will result in loss of control of sponsorship funds

Severity: Minor

In the `UpdateSponsor`, `DeleteSponsor`, and `WithdrawSponsorFunds` functions in `x/sponsor-contract-tx/keeper/msg_server.go:117,206,269`, all sponsorship operations require contract admin validation. If the contract admin were to be cleared using `ClearAdmin`, the existing sponsorship would be immutable.

This results in permanent loss of control over sponsor funds, which remain locked in the sponsor address indefinitely.

Even in the case of an admin update, the keeper has no hook or reconciliation path to update `CreatorAddress` in response to wasm admin mutations, creating admin divergence and potential orphaned sponsorship data that cannot be managed by any party.

Recommendation

We recommend handling the case where the contract admin is cleared or changed. This can be accomplished in one of the following ways. Consider wrapping wasmd `ClearAdmin` and `UpdateAdmin` messages and handling the changes. Or more simply, keeping the initial address that was used to create the sponsorship rather than querying the admin for these sponsorship operations.

Status: Resolved

5. Multiple `MsgExecuteContract` allowed without a cap in sponsored transactions

Severity: Minor

In `x/sponsor-contract-tx/ante/ante.go:526-560`, the `validateSponsoredTransaction` function ensures all `MsgExecuteContract` messages target the same contract but does not impose a cap on the number of `MsgExecuteContract` messages per transaction.

This allows a sponsored transaction to bundle an arbitrarily large number of executions to the same contract, potentially increasing verification and policy-check workload, bloating ante processing, and raising DoS risk or unfair resource consumption by sponsored users.

Recommendation

We recommend introducing a reasonable upper bound on the number of `MsgExecuteContract` messages permitted in a single sponsored transaction, returning a skip/reject when exceeded, and documenting the limit in module parameters for configurability.

Status: Resolved

6. Incomplete genesis state validation leading to potential state inconsistencies

Severity: Minor

The `ValidateGenesis` function in `x/sponsor-contract-tx/types/types.go` performs only partial validation of the `GenesisState` structure.

While it checks for duplicate sponsors and empty addresses, it omits several validation steps.

Specifically, the function does not verify that `ContractAddress`, `CreatorAddress`, and `SponsorAddress` are valid addresses, nor does it ensure that `ContractAddress` exists within a valid contract and that `SponsorAddress` is derived from it.

Additionally, temporal consistency between `CreatedAt` and `UpdatedAt` fields is not enforced (e.g., `CreatedAt` should be less than or equal to `UpdatedAt`).

Missing checks for non-empty values, such as `MaxGrantPerUser`, may allow improperly initialized genesis configurations. Similar omissions exist for the `UserGrantUsages` structure, potentially leading to invalid or inconsistent genesis states.

Recommendation

We recommend enhancing the `ValidateGenesis` function to enforce comprehensive validation of all genesis parameters.

Status: Resolved

7. Unsafe error formatting with raw user addresses enables log amplification and log injection

Severity: Minor

The `AnteHandle` constructs error messages that interpolate unsanitized, user-supplied addresses directly into log entries and API error responses, such as in calls to `errorsmod.Wrapf(..., "contract address %s not found", contractAddr)`.

When address parsing fails, the full input string, including potentially very large payloads, is echoed back to both clients and logs.

This behavior allows an attacker to submit excessively long or specially crafted address values containing control characters, newlines, or escape sequences. As a result, the system becomes vulnerable to response amplification, excessive log growth, and log injection attacks.

Additionally, since the transaction fails and the sender has no funds, no gas will be charged to the attacker, making this vector inexpensive to abuse.

Recommendation

We recommend validating the contract address in the `validateSponsoredTransaction` function before proceeding with the rest of the execution.

Status: Resolved

8. Sponsorship creation enables abuse

Severity: Minor

There are multiple factors in the `SetSponsor` function in `x/sponsor-contract-tx/keeper/msg_server.go:74-90` that allow for malicious actors to create sponsored contracts that may be utilized to create spam and even potentially degrade the performance of the chain.

In the `SetSponsor` function in `x/sponsor-contract-tx/keeper/msg_server.go:74-90`, the sponsor address is derived from the sponsored contract address, but there is no fee transfer during setup. Because the fee transfer occurs separately, this creates a situation where an attacker can cheaply create a very large amount of sponsored contracts. The only economic barrier is contract creation, and transaction fees.

This can be further compounded by the fact that there is no validation to ensure that `MaxGrantPerUser` is sufficient to cover the minimum transaction fees required by validators. A malicious sponsor can set `MaxGrantPerUser` to a very low amount while validators require higher minimum fees.

This can be used to create a more wide-scale impact on the chain where an attacker can create one or many sponsorship contracts, with the policy query optimized to use near `params.MaxGasPerSponsorship` that will then always error when the fee is checked against the user maximum. The end user will still pay the gas for the failed transaction.

Recommendation

We recommend ensuring that, in the `SetSponsor` function, some base amount of funds are transferred into the sponsored address. Define this minimum as a state variable, and also make sure to consider this minimum when allowing the sponsor fund withdrawal.

Status: Resolved

9. Unsafe marshalling of policy query payloads

Severity: Minor

In `x/sponsor-contract-tx/keeper/keeper.go:86-99` the keeper constructs the `PolicyQueryJSON` (the `queryMsg` map) and calls `json.Marshal` before entering any gas-limited execution context.

Constructing large or deeply-nested JSON payloads (or re-marshalling already-large msg_data blobs) lets a malicious sender force significant CPU and memory work on a node during ante/keeper processing.

An attacker can craft transactions with many complex or deeply nested contract messages so the node performs expensive `json.Marshal` operations for each message.

Due to the nature of the sponsorship ante handler, the failed transaction would not result in any gas charged to the sender.

Recommendation

We recommend validating and bounding the size and complexity of any untrusted JSON payload before constructing or marshalling `queryMsg`.

Status: Resolved

10. Redundant user balance check in sponsored fee payment section

Severity: Informational

In the `AnteHandle` function in `x/sponsor-contract-tx/ante/ante.go:380-407`, there is a user balance check that is redundant due to an earlier identical check at `lines 197-225`. The earlier check already determines if the user has sufficient balance to pay fees themselves and returns early if they do, making the later check in the sponsored fee payment section redundant. This creates dead code that will never be executed and adds unnecessary complexity to the codebase.

Recommendation

We recommend removing the redundant user balance check at `lines 380-407` since the earlier check at `lines 197-225` already handles this case and prevents execution from reaching the later section when the user can self-pay.

Status: Resolved

11. Query all-sponsors lacks pagination support

Severity: Informational

In the `GetCmdQueryAllSponsors` function in `x/sponsor-contract-tx/client/cli/query.go:34-58`, the `all-sponsors` query returns all sponsor addresses without pagination support. The query uses an empty `QueryAllSponsorsRequest` and directly returns all results without any pagination parameters. As the number of sponsors grows, this query will become increasingly slow and resource-intensive, and will cause a poor user experience.

Recommendation

We recommend implementing pagination support for the all-sponsors query.

Status: Resolved

12. `SponsorshipEnabled` check should occur at the beginning of the antehandle to avoid unnecessary computation

Severity: Informational

In the `AnteHandle` function in `x/sponsor-contract-tx/ante/ante.go:149-167`, the global `SponsorshipEnabled` parameter check occurs after contract validation, sponsor lookup, and other processing. This causes unnecessary computation when sponsorship is globally disabled, including contract existence validation, sponsor retrieval, user address determination, and fee validation. This wastes gas and processing time for transactions that will ultimately be rejected due to the global toggle.

Recommendation

We recommend moving the SponsorshipEnabled check to the beginning of the AnteHandle function, immediately after the feegrant check and before any contract-specific validation or processing.

Status: Resolved

13. The `getUserAddressForSponsorship` is called multiple times for the same transaction causing redundant computation

Severity: Informational

In the `AnteHandle` function in `x/sponsor-contract-tx/ante/ante.go:118, 169, and 480`, the `getUserAddressForSponsorship` function is called three times in different code paths for the same transaction. This function performs expensive operations including message signer validation, consistency checks across all messages, and FeePayer validation.

Since the transaction and its signers don't change during the ante handler execution, calling this function multiple times results in redundant computation and wasted gas. The user address should be determined once at the beginning of the function and reused throughout the different code paths.

Additionally, this function performs multiple iterations that can introduce additional processing time and overhead for transactions with multiple messages.

Recommendation

We recommend calling `getUserAddressForSponsorship` once at the beginning of the `AnteHandle` function, after the initial validation checks, and storing the result in a variable that can be reused throughout the function.

Status: Resolved

14. Replace hardcoded denom name

Severity: Informational

In the `AnteHandle` function in `x/sponsor-contract-tx/ante/ante.go:200`, the fee denomination is hard-coded as `peaka` in the validation logic. This hardcoded string appears throughout the codebase across multiple files.

Using a hardcoded string for the denomination makes the code less maintainable and more prone to errors if the denomination needs to be changed in the future. It also violates the DRY principle and makes it difficult to update the denomination across the entire codebase consistently.

Recommendation

We recommend defining a constant for the supported denom in a central location, such as the types package, and replacing all hardcoded `peaka` strings throughout the codebase with this constant.

Status: Resolved

15. Unused error `ErrInvalidPolicyResponse`

Severity: Informational

The `ErrInvalidPolicyResponse` constant in `x/sponsor-contract-tx/types/errors.go:17` is defined but never referenced anywhere in the module. It is best practice to remove all unused error types.

Recommendation

We recommend removing or using the `ErrInvalidPolicyResponse` error type.

Status: Resolved