Security Audit Report

1Matrix

v0.1

January 8, 2026

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

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 1Matrix Joint Stock Company to perform a security audit of Security audit of the 1Matrix custom Cosmos SDK modules.

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/OneMatrixL1/1matrix-fullnode/
Commit	`7b3d17ae9d51f9488e21bb70acd56d0c96f8318d`
Scope	The following modules were in scope along with their integration into the Cosmos SDK application: - `admin` - `blacklist` - `whitelist`

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 audit scope covers the admin, blacklist, and whitelist modules of a permissioned Cosmos SDK blockchain for Vietnam Blockchain Services Network (VBSN) that adds access controls. The admin module provides custom governance where admin accounts submit and vote on proposals. The blacklist module blocks blacklisted addresses from initiating transactions or receiving tokens, enforced via ante handlers. The whitelist module restricts staking operations like validator creation to whitelisted addresses only.

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	Low-Medium	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	-
Test coverage	Medium-High	The client has provided a custom test coverage report that includes only business logic and omits generated code. The filtered coverage of 71.3% represents the actual test coverage of custom business logic. Admin Module: Governance and administrative controls (87.0% keeper, 82.1% CLI) Blacklist Module: Address restriction enforcement (100.0% ante, 88.8% keeper) Whitelist Module: Permissioned access control (92.0% ante, 85.2% keeper)

Summary of Findings

No	Description	Severity	Status
1	Authz and Group messages can bypass message checks	Critical	Resolved
2	EVM extension ante handler is missing critical security checks	Critical	Resolved
3	Ethereum transactions bypass blacklist restrictions	Critical	Resolved
4	Outdated dependencies carry the risk of vulnerabilities	Major	Resolved
5	Blacklist and whitelist ante decorators positioned before gas decorator	Major	Resolved
6	Blacklisted and non-whitelisted users can retain staked tokens through redelegations	Major	Resolved
7	Blacklist module has empty `Params` struct but `UpdateParams` handler exists	Minor	Resolved
8	`WhitelistAnteDecorator` incorrectly restricts `MsgBeginRedelegate`	Minor	Resolved
9	Missing parameter validation allows invalid module configuration	Minor	Resolved
10	No limit on proposal message count can cause execution failures	Minor	Resolved
11	Incorrect error message references wrong module	Informational	Resolved
12	Event message string starts with leading comma	Informational	Resolved
13	Duplicate admin indices possible due to deletion without reindexing	Informational	Resolved
14	Remove commented code blocks	Informational	Resolved

Detailed Findings

1. Authz and Group messages can bypass message checks

Severity: Critical

In the `AnteHandle` function in `x/whitelist/ante/whitelist_ante.go:32-37`, the code only checks top-level messages via `tx.GetMsgs`, which does not unwrap nested messages inside `MsgExec`. When a non-whitelisted user wraps a restricted staking message inside `MsgExec`, the ante handler sees only `MsgExec`, which doesn't match the staking pattern, so the whitelist check is skipped. The nested staking message is then executed on behalf of the granter (who may be whitelisted) without validation, allowing unauthorized staking operations. This bypass is also possible with the `group` module.

The same issue is present in the `AdminGovSwitchDecorator` in `app/ante.go:162`.

Recommendation

We recommend checking nested messages inside `MsgExec` by detecting `MsgExec` messages, extracting their nested messages, and applying the same validation to those nested messages. This ensures that whitelist and admin governance restrictions cannot be bypassed through the authz module. Alternatively, a higher level restriction can be placed on the whitelist restriction to prevent non-whitelisted addresses from these nested message bypasses by disallowing `authz` and `group` message types.

Status: Resolved

2. EVM extension ante handler is missing critical security checks

Severity: Critical

In `app/ante.go:77`, the `newMonoEVMAnteHandler` function only implements a blacklist check for transactions with the "`/cosmos.evm.vm.v1.ExtensionOptionsEthereumTx`" extension option. This handler is missing all critical security validations that are standard in EVM transaction processing, including signature verification, nonce validation, gas limit checks, fee deduction, and sender authorization.

When a transaction includes the EVM extension option, it bypasses the comprehensive `newCosmosAnteHandler` chain and instead routes through `newMonoEVMAnteHandler`, which only validates whether the sender is blacklisted.

This leads to multiple severe security vulnerabilities:

- No signature verification: attackers can submit transactions without valid signatures
- No fee deduction: transactions can be executed without paying gas fees

- No nonce validation: replay attacks and transaction ordering issues are possible
- No gas limit enforcement: unlimited computational resources can be consumed
- No sender authentication: transaction sender claims could be incorrect

This represents a complete bypass of the transaction security model for EVM-type transactions. An attacker can exploit this to execute arbitrary transactions without proper authorization, drain funds, perform replay attacks, and cause denial-of-service by consuming validator resources without payment.

Recommendation

We recommend implementing a complete EVM ante handler chain that includes all necessary security checks. For example, reference the EVMOS implementation as a baseline.

Status: Resolved

3. Ethereum transactions bypass blacklist restrictions

Severity: Critical

In the `NewAnteHandler` function in `app/ante.go:123-125`, when a transaction has the `ExtensionOptionsEthereumTx` extension option, it routes to `newMonoEVMAnteHandler` which re-uses the same `NewCheckBlacklistDecorator` designed specifically for cosmos addresses and is not EVM compatible. This allows blacklisted EVM users to execute messages and bypass the blacklist all together.

The `NewCheckBlacklistDecorator` validates that new blacklist addresses are bech32 compatible which would reject any EVM addresses from being added to the blacklist at all.

Recommendation

We recommend implementing a purpose built blacklist handler for EVM transactions.

Status: Resolved

4. Outdated dependencies carry the risk of vulnerabilities

Severity: Major

The application contains multiple known vulnerabilities across dependency modules that directly affect the codebase. The most critical issues include:

- CometBFT v0.38.17 (`GO-2025-4025`): Invalid `BitArray` handling could lead to network halt.

- Cosmos SDK v0.53.2 (GO-2025-3803): Integer overflow in the validator rewards pool could allow a malicious actor to halt the chain.
- Cosmos SDK v0.53.2 (GO-2023-1881, GO-2023-1821): Two medium-severity issues in the `x/crisis` module, where the `ConstantFee` that attempts to prevent spam attacks is not charged properly, and a chain halt will not occur when users submit violated invariants to the `x/crisis` module.

These vulnerabilities pose significant risks, potentially affecting the network's availability and consensus mechanism.

Recommendation

We recommend updating all dependencies to their latest stable versions to ensure the application benefits from the latest security patches, bug fixes, and performance improvements.

Status: Resolved

5. Blacklist and whitelist ante decorators positioned before gas decorator

Severity: Major

In `app/ante.go:49-50`, the blacklist and whitelist ante decorators are positioned at the beginning of the ante handlers, which is called before the critical gas-related decorators such as `NewConsumeGasForTxSizeDecorator` and `NewDeductFeeDecorator`. This creates a potential denial-of-service (DoS) vector that allows attackers to force validators to perform computational work without paying the required gas fees.

Specifically, both decorators perform state reads and iterations that consume validator resources. The `CheckBlacklistDecorator` iterates through all transaction signers and performs keeper lookups for each signer against the blacklist state. The `WhitelistAnteDecorator` uses regex pattern matching against all transaction messages, retrieves signers, and performs keeper lookups for each signer when `MsgCreateValidator` and `MsgBeginRedelegate` messages are detected.

An attacker can exploit this by submitting transactions with multiple signers to inflate the iteration cost, sending large volumes of transactions targeting blacklisted/non-whitelisted addresses, and forcing validators to perform expensive state reads and regex operations without paying gas. At scale, this attack vector could significantly degrade validator performance, potentially leading to consensus delays and a chain halt.

Recommendation

We recommend repositioning both the `CheckBlacklistDecorator` and `WhitelistAnteDecorator` to execute after gas-related decorators. Specifically, they should be placed after `NewConsumeGasForTxSizeDecorator`,

`NewDeductFeeDecorator`, and `NewSigGasConsumeDecorator` to ensure that base gas is charged for transaction size, fees are deducted, and signature verification gas is charged before any expensive blacklist or whitelist validation operations are performed.

Status: Resolved

6. Blacklisted and non-whitelisted users can retain staked tokens through redelegations

Severity: Major

In `x/blacklist/keeper/blacklisted_addresses.go:137`, when a user is blacklisted, the `MsgAddBlacklistedAddress` message calls `UndelegateBlacklistedAddress` to undelegate all their staked positions. The intended behavior is that blacklisted and non-whitelisted users should not be allowed to have any delegations to validators.

However, the function fails to handle redelegations, allowing blacklisted users to retain staked tokens through active redelegation entries. Redelegations represent tokens being moved from one validator to another and remain locked during the redelegation period. Since redelegations are not handled, a blacklisted user can maintain staked positions and continue earning rewards through active redelegations.

This same issue exists in the `undelegateUnWhitelistAddress` function in `x/whitelist/keeper/msg_whitelist.go:105`, where non-whitelisted users who are removed from the `x/whitelist` module can similarly retain staked tokens through redelegations.

Consequently, this undermines the enforcement mechanisms of both the `x/blacklist` and `x/whitelist` modules, as users can circumvent forced undelegations by having active redelegations at the time of blacklisting or whitelist removal.

Recommendation

We recommend retrieving and undelegating all redelegations for the blacklisted or non-whitelisted address using the `GetRedelegations` method from the `x/staking` keeper.

Status: Resolved

7. Blacklist module has empty `Params` struct but `UpdateParams` handler exists

Severity: Minor

In the `UpdateParams` method in `x/blacklist/keeper/msg_server.go:125-136`, the handler exists but the `Params` struct is empty, making the method, its proto definition,

and related keeper methods (`GetParams`, `SetParams`) unnecessary. This adds code bloat and can cause confusion.

Recommendation

We recommend removing the `UpdateParams` handler from the module's message server along with its corresponding functions that are unused code.

Status: Resolved

8. `WhitelistAnteDecorator` incorrectly restricts `MsgBeginRedelegate`

Severity: Minor

The `WhitelistAnteDecorator` in `x/whitelist/ante/whitelist_ante.go:29` includes `MsgBeginRedelegate` in its regex pattern, restricting non-whitelisted users from executing this particular staking message.

According to the business logic, the whitelist module is intended to only prevent non-whitelisted accounts from creating validators via `MsgCreateValidator`, while permitting all other `x/staking` module interactions.

However, the current regex pattern that is "*^/cosmos\.staking\..*\.(MsgCreateValidator|MsgBeginRedelegate)$*" blocks both `MsgCreateValidator` and `MsgBeginRedelegate`, which contradicts the intended business logic. The restriction on `MsgBeginRedelegate` prevents non-whitelisted users from moving their delegations between validators.

Recommendation

We recommend removing `MsgBeginRedelegate` from the regex pattern to align with the intended business logic.

Status: Resolved

9. Missing parameter validation allows invalid module configuration

Severity: Minor

In the `Validate` function in `x/admin/types/params.go:44-45`, the method returns `nil` without validating any parameter fields. This impacts both genesis initialization and parameter updates. Invalid parameters can be set, including: `VotingThreshold`, `VotingPeriod`, and `MinAdminNumber`. These invalid values are only detected at runtime when used and would cause errors during tally calculation. This issue is also found in the whitelist module's param validation in `x/whitelist/types/params.go:14`.

In addition, during the admin module's `InitGenesis` in `x/admin/module/genesis.go:12` `genState.AdminList` is not validated to ensure that the addresses are valid before setting them in the store.

Recommendation

We recommend implementing parameter validation for the admin and whitelist modules' parameters and during `initGenesis` to ensure that the `genState.AdminList` contains valid admin addresses.

Status: Resolved

10. No limit on proposal message count can cause execution failures

Severity: Minor

In the `SubmitProposal` function in `x/admin/keeper/proposal.go:76-92`, the code processes all messages without enforcing a maximum count. A proposal with an extremely large number of messages can pass voting but fail during execution due to gas limits, block size constraints, or timeout issues, wasting governance resources and voter time on proposals that cannot execute.

Recommendation

We recommend adding a maximum message count limit and validating a proposal's message count is below the limit before proposal creation, rejecting proposals that exceed the limit with an appropriate error.

Status: Resolved

11. Incorrect error message references wrong module

Severity: Informational

The `safeExecuteHandler` function in `x/admin/module/module.go:413` contains an error message that incorrectly references "*x/gov*" instead of "*x/admin*" when a proposal message execution panics during the `EndBlock` process.

This creates confusion during debugging and incident response, as developers may look at the wrong module when investigating panics during admin proposal execution.

Recommendation

We recommend updating the comment to reflect the `x/admin` module.

Status: Resolved

12. Event message string starts with leading comma

Severity: Informational

In the `SubmitProposal` function in `x/admin/keeper/proposal.go:77`, the code builds `msgsStr` by appending messages with `fmt.Sprintf(",%s", ...)`, which results in a string starting with a comma. While this doesn't cause functional issues, it creates malformed event attribute values and could cause parsing issues in off-chain indexers or monitoring tools that expect properly formatted comma-separated values.

Recommendation

We recommend initializing `msgsStr` as empty and only adding commas between messages to ensure proper formatting of the event attribute.

Status: Resolved

13. Duplicate admin indices possible due to deletion without reindexing

Severity: Informational

In the `AddAdmin` function in `x/admin/keeper/msg_server.go:174-182`, new admins are assigned an `Index` using `TotalAdminAddresses` + 1. When admins are deleted via `RemoveAdmin`, their indices are not reassigned, and subsequent additions reuse indices. This can result in multiple admins sharing the same Index value. Since Index is stored as metadata and not used for lookups, storage keys, or vote threshold calculations, this does not cause functional issues. However, it violates the expectation that `Index` represents a unique sequential identifier and could cause confusion in queries, logs, or future code that assumes Index uniqueness.

Recommendation

We recommend either removing or renaming the `Index` field if it's not used for unique sequential indexing, or implementing a reindexing mechanism when admins are deleted to maintain sequential uniqueness. Alternatively, document that Index is informational only and may contain duplicates after deletions.

Status: Resolved

14. Remove commented code blocks

Severity: Informational

There are multiple occurrences of commented code blocks in the audit scope. For example in `x/blacklist/ante/check_blacklist.go:21`, validation is commented out. This impacts the readability and maintainability of the codebase. It is best practice to remove commented code blocks if they are not intended to be implemented.

Recommendation

We recommend removing commented codeblocks from the codebase.

Status: Resolved