Audit Report

Validator-scheduled Standard Cosmos Hard Fork

v1.0

December 16, 2024

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 TC Management Company Limited to perform a security audit of the Validator-scheduled Standard Cosmos Hard Fork feature.

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://gitlab.com/thorchain/thornode
Commit	`f04a8616804d345d24c9d477c2ef3d92b822ad89`

Scope	The scope was restricted to the changes compared to commit `37bb933c2bcabf8af47c82bc36ebf78a54ea1147` in the following directory:
	`.`
	`└── x`
	`    └── thorchain`

Fixes verified at commit	`11d1915af2e4e0572e485dfe820b2cde8100c615`

	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

THORChain is a decentralized cross-chain liquidity protocol that enables users to swap assets across different blockchain networks without relying on centralized exchanges or custodial intermediaries. It facilitates seamless and secure trading of native cryptocurrencies like Bitcoin, Ethereum, Binance Coin, and others directly from users' wallets.

The scope of this audit is restricted to the updates related to the validator-scheduled on-chain upgrades feature using the upgrade module.

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	-
Level of documentation	Medium-High	Detailed documentation is available in the merge request. Additionally, extensive comments are present explaining most of the critical functions and logic.
Test coverage	Medium-High	-

Summary of Findings

No	Description	Severity	Status
1	Unrestricted proposal creation could lead to Denial of Service	Major	Resolved
2	The `queryUpgradeProposals` query might fail for too many proposals	Minor	Acknowledged
3	Upgrade proposals for future version can be blocked	Minor	Acknowledged
4	Missing error when voting multiple times on the same proposal	Informational	Acknowledged
5	Validators can reject non-existing upgrades	Informational	Resolved
6	Upgrade binary hash format is not enforced	Informational	Acknowledged
7	Usage of magic numbers decreases maintainability	Informational	Acknowledged
8	Code quality improvements	Informational	Acknowledged

Detailed Findings

1. Unrestricted proposal creation could lead to Denial of Service

Severity: Major

`x/thorchain/keeper/v1/keeper_upgrade.go` contains logic responsible for proposal handling, such as its creation, voting, or removal. Every validator can create a new upgrade proposal, using the `MsgProposeUpgrade` message, with some specific `Upgrade.Info`, `Name`, and `Height`.

Proposals are then stored based on their key, which is represented by a predefined prefix and name provided by the validator. This is problematic, because there is no validation whether the caller does not already have an active proposal. If the name of the proposal is unique, the validator has an opportunity to create multiple proposals without restriction.

The issue occurs when the `RemoveExpiredUpgradeProposals` function, which is called in the `BeginBlocker`, is iterating over all proposals to find expired ones. If the amount of proposals is significant, this `for` loop might slow down block production which could eventually halt the chain, leading to a Denial of Service.

We classify this issue as major instead of critical, as only validators can create proposals, who are assumed to not benefit from this kind of attack.

Recommendation

We recommend implementing a mechanism that prevents the creation of an unlimited number of proposals by a single validator. If it is desirable that one validator should be able to create multiple proposals, we recommend limiting the number of proposals per validator, for example to 5.

Status: Resolved

2. The `queryUpgradeProposals` query might fail for too many proposals

Severity: Minor

The `for` loop used by the `queryUpgradeProposals` function in `x/thorchain/querier.go:2563` may not work properly when the number of proposals is too large.

After loading many objects into memory and then iterating one by one, it is possible that the gas upper limit will be reached and the query may fail.

Recommendation

We recommend implementing a mechanism that prevents the creation of an unlimited number of proposals, or to implement pagination for the `queryUpgradeProposals` function.

Status: Acknowledged

3. Upgrade proposals for future version can be blocked

Severity: Minor

Within `x/thorchain/types/msg_upgrade.go:37-39`, the `ValidateBasic` function validates that the name used in the upgrade proposal is a version tag composed according to the Semantic Versioning rules.

A malicious validator can block upgrade proposals related to future versions by creating preventive bogus proposals. Any versions used in such proposals will be blocked until the proposals are expired and pruned. Additionally, the malicious validator can set height arbitrarily far in the future and block specific versions forever.

Developers could mitigate such an attack to some extent, by incrementing “patch” number, or other parts of the version tag. However, this could disorganize the development process.

Recommendation

We recommend implementing a limit for the block height and charging a fee for upgrade proposals. Additionally, slashing and temporary eviction from the active validator set could be considered.

Status: Acknowledged

4. Missing error when voting multiple times on the same proposal

Severity: Informational

Within `ApproveUpgradeHandler`, if a validator votes again on an upgrade proposal using an already cast vote, for example "Approve", he is not informed about the fact that no changes were made to the chains’ state.

The vote is overwritten with an identical one, and then an event is emitted informing about the correct vote count. While this does not consitute a security issue, it could confuse validators and potential other off-chain components.

Furthermore, if off-chain aggregators use logs and events for certain statistical operations - they may be inaccurate due to the recording of no-op operations.

Recommendation

We recommend validating whether the validator has already voted for a specific upgrade proposal. If so, and if this vote is of the same type, then an appropriate error message should be returned.

Status: Acknowledged

5. Validators can reject non-existing upgrades

Severity: Informational

In `x/thorchain/handler_upgrade.go:247-261`, the `RejectUpgradeHandler` message does not validate that the upgrade to reject exists. Ideally, validators would only reject proposed upgrades to stop them from being scheduled.

Although we did not find a security vulnerability introduced by this issue, allowing validators to reject non-existing upgrades does not present a use case and may have unforeseen consequences in the future.

Recommendation

We recommend validating that the upgrade exists.

Status: Resolved

6. Upgrade binary hash format is not enforced

Severity: Informational

In `proto/thorchain/v1/x/thorchain/types/msg_upgrade.proto:9-12`, the `Upgrade` structure is defined. Its `info` field is supposed to point to the binary necessary to be installed by a node to consider the upgrade performed. An off-chain agreement could be that the `info` field contains a URL to the binary and its hash or checksum that could be checked by each node after downloading the binary.

However, in case a proposal does not follow such an agreed-upon approach, or if the hash is incorrect, there is a risk of discovering it only after the proposal approval.

Recommendation

We recommend extending the `Upgrade` structure with an explicit field containing the hash of the binary, as well as specifying what hash function should be used and validating the size of the hash value.

Status: Acknowledged

7. Usage of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `x/thorchain/ante.go:126`
- `x/thorchain/handler_swap.go:48`
- `x/thorchain/handler_upgrade.go:58, 159, 229`
- `x/thorchain/handler.go:67`
- `x/thorchain/helpers.go:144`
- `x/thorchain/managers.go:390, 464`
- `x/thorchain/aggregators/dex_mainnet.go:12`
- `x/thorchain/keeper/v1/keeper_upgrade.go:69,160`
- `x/thorchain/querier.go:2644`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Acknowledged

8. Code quality improvements

Severity: Informational

In the following instances, the code quality can be improved:

- The storage key construction in `x/thorchain/keeper/v1/keeper_upgrade.go:34` and `49` are identical. To reduce code duplication, consider creating a function that calls `fmt.Sprintf("%s%s", prefixUpgradeProposals, name)` and use it in both of these places.
- In `x/thorchain/keeper/v1/keeper_upgrade.go:164`, the `prefix` variable is constructed every time in the loop based on the proposal votes. This is unnecessary because the variable will be the same across all iterations. Consider defining the `prefix` variable outside of the loop to decrease computational complexity.

Recommendation

We recommend applying the aforementioned recommendations.

Status: Acknowledged