CoinFabrik

THORChain Audit

January 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the THORChain project.

During this audit we found no critical issues, no high severity issues, one medium issue and no minor issues. Also, one enhancement was proposed.

Scope

The audited files are from the git repository located at https://gitlab.com/thorchain/thornode. The audit is based on the commit **`2a62b35a86d41a612b834f77c662e73b2d630de0`**.

The scope for this audit includes and is limited to the following files:

- `app/`: Node implementation.
- `chain/avalanche/src/contracts/`: Avalanche router.
- `chain/ethereum/contracts/`: Ethereum router and tokens.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation.

Methodology

CoinFabrik was provided with the source code. Our auditors spent three weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters

- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
ME-01	Regtest App Denial Of Service	Medium	Unresolved
MI-01	Outdated Version of Solidity Compiler	Minor	Unresolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

No issues found.

High Severity Issues

No issues found.

Medium Severity Issues

ME-01 Regtest App Denial Of Service

Location:

- `app/app_regtest.go:30`

Classification:

- CWE-400: Uncontrolled Resource Consumption[1]

The `app_regtest` application creates a test node and opens an HTTP web server to unblock every block generation. But this webserver does not have authentication, allowing anybody in the network to unlock a block generation. Additionally, as the web server also lacks a timeout, it is possible for a single client to lock all block generations by never closing the http connections. This would only affect the regtest app, so the impact is lowered to medium.

Recommendation

Implement timeouts and optionally, simple authentication so only authorized clients can connect to the web server, if the node is used in a shared or public environment.

[1]https://cwe.mitre.org/data/definitions/400.html

Status

Unresolved

Minor Severity Issues

MI-01 Outdated Version Of Solidity Compiler

Location:

- `chain/ethereum/contracts/eth_rune.sol:2,`
- `chain/ethereum/contracts/evilToken.sol:2,`
- `chain/ethereum/contracts/Token.sol:2,`
- `chain/ethereum/contracts/USDT.sol:2`

Classification:

- CWE-1395: Dependency on Vulnerable Third-Party Component[2]

The above contracts use old or very old Solidity compiler software. Newer compiler versions implement security fixes and more safe features. For example, safemath code is obsolete and should be upgraded at USDT.sol, by upgrading it to the latest version of the Solidity compiler.

Recommendation

Upgrade all contracts to use the latest version of the Solidity compiler available.

Status

Unresolved

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Use OpenZeppelin's Standard Libs	Not implemented

[2]https://cwe.mitre.org/data/definitions/1395.html

EN-01 Use OpenZeppelin's Standard Libs

Location:

- `chain/avalanche/src/contracts/AvaxRouter.sol:75`,
- `chain/avalanche/src/contracts/AvaxAggregator.sol:30`,
- `chain/ethereum/contracts/THORChain_Aggregator.sol:32`,
- `chain/ethereum/contracts/THORChain_Router.sol:52`,
- `chain/ethereum/contracts/eth_rune.sol:51`

The above contracts use a custom reentrancy guard. While this implementation is correct, a more standard way to do it is to inherit from OpenZeppelin standard `ReentrancyGuard.sol`.

A similar enhancement can be done by using OpenZeppelin `Ownable.sol` instead of a custom ownable class at `eth_rune.sol`.

Recommendation

Inherit from OpenZeppelin standard `ReentrancyGuard.sol` and `Ownable.sol`.

Status

Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Upgrades

The USDT contract uses a custom upgrade mechanism.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

USDT.sol

Owner

The owner role can: Transfer the ownership of the contract, pause and unpause the contract, blacklist and un-blacklist users, burn blacklisted funds, deprecate the contract address, mint and redeem tokens and set token parameters.

eth_rune.sol

Owner

The owner role can: Transfer and renounce the ownership of the contract, and mint tokens.

Changelog

- 2024-01-26 – Initial report based on commit `2a62b35a86d41a612b834f77c662e73b2d630de0`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the THORChain project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.