Security Audit Report

Hippocrat Hippo Protocol

v1.0

March 21, 2025

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 Hippocrat DAO Foundation to perform a security audit of Hippocrat Hippo Protocol.

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/hippocrat-dao/hippo-protocol
Commit	`4f464e5875c7d03e85cd0d3643f8b0f43da0f288`
Scope	The entire repository was in scope except for the `sdk` folder.
Fixes verified at commit	`164c99d86da0bbc5189fcff61742156d6bee795c` 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 Hippo Protocol is an application-specific blockchain built using the Cosmos SDK, designed for decentralized management of healthcare data.

The protocol implements a custom minting function for its tokens, allowing for controlled issuance and distribution within its ecosystem.

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	Comments describe most of the architectural design choices.
Test coverage	Low ➔ Medium-High	Tests were not present at the audited commit. The client has increased the test coverage during the engagement to `66.10%`.

Summary of Findings

No	Description	Severity	Status
1	Infinite gas allocated per block allows attackers to halt the chain	Critical	Resolved
2	Insufficient `MaxAgeDuration` and `MaxAgeNumBlocks` values may allow malicious validators to escape penalties	Major	Resolved
3	Usage of deprecated `x/crisis` module allows attackers to execute DoS attacks	Major	Resolved
4	Oversized maximum block size could allow DoS attacks	Major	Resolved
5	Vulnerabilities in outdated dependencies	Major	Resolved
6	Misconfigured wallet coin type leading to compatibility issues	Minor	Acknowledged
7	Incorrect `bech32` prefix in account keeper	Minor	Resolved
8	CLI discards filters during genesis export	Informational	Resolved
9	Misplaced configuration of initialization parameters	Informational	Resolved
10	Redundant definition of `MakeEncodingConfig` function	Informational	Resolved
11	Use of magic numbers decreases maintainability	Informational	Resolved

Detailed Findings

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

Severity: Critical

In `hippod/cmd/init.go:152`, the `ConsensusParams.Block` field is defined in the `DefaultBlockParams` function, which sets the `MaxGas` field to `-1`.

In the context of the Cosmos SDK `baseapp` it leads to the usage of an `infiniteGasMeter` during block handling. Please 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 `MaxGas` 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: Resolved

2. Insufficient `MaxAgeDuration` and `MaxAgeNumBlocks` values may allow malicious validators to escape penalties

Severity: Major

In `hippod/cmd/init.go:248-249`, the `MaxAgeDuration` and `MaxAgeNumBlocks` fields for the consensus parameters are set to the unbonding period (21 days) and the number of estimated blocks (unbonding period/block time per second).

However, these fields may not be sufficient to provide coverage for the entire unbonding period for the chain, as mentioned in `GHSA-555p-m4v6-cqxv`.

Consequently, 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 applying the following recommendations:

- The `MaxAgeDuration` parameter should be updated to exceed the chain's unbonding period.
- The `MaxAgeNumBlocks` parameter should be updated to exceed the number of estimated blocks that will be produced by the chain throughout the unbonding period.

Status: Resolved

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

Severity: Major

The chain 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 chain configuration. Instead, simulation tests should be enhanced and the implementation of the `x/circuit` module should be considered.

Status: Resolved

4. Oversized maximum block size could allow DoS attacks

Severity: Major

In `hippod/cmd/init.go:152`, the `ConsensusParams.Block` field is defined in the `DefaultBlockParams` function, which sets the `MaxBytes` field to `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 `MaxBytes` 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: Resolved

5. Vulnerabilities in outdated dependencies

Severity: Major

In the `go.mod` file, vulnerable versions of dependencies are used, which may allow attackers to exploit and impact the chain's availability, integrity, and confidentiality:

- `github.com/cometbft/cometbft@v0.37.5`
  - GHSA-r3r4-g7hq-pq4f
  - GHSA-22qq-3xwm-r5x4
  - GHSA-g5xx-c4hv-9ccc
  - GHSA-hq58-rf2h-6rr7
- `github.com/cosmos/cosmos-sdk@v0.47.12`
  - GHSA-8wcc-m6j2-qxvm
- `cosmossdk.io/math@v1.3.0`
  - GHSA-7225-m954-23v7

Recommendation

We recommend updating the dependencies to `github.com/cometbft/cometbft@v0.38.17`, `github.com/cosmos/cosmos-sdk@v0.47.15`, and `cosmossdk.io/math@v1.4.0`.

Status: Resolved

6. Misconfigured wallet coin type leading to compatibility issues

Severity: Minor

In `types/consensus/wallet.go:16-27`, the `SetWalletConfig` function modifies the wallet's configuration before sealing it. Specifically, it sets the CoinType used for hierarchical deterministic wallets to `BIP44CoinType`, which is hardcoded to `0`. The `CoinType` is a crucial parameter in constructing the wallet's derivation path:

```
m / purpose' / coin_type' / account' / change / address_index
```

However, `CoinType 0` is reserved for Bitcoin, as per the standard list of coin types defined in SLIP-0044.

Consequently, this misconfiguration could cause third-party wallet applications to misinterpret the Cosmos-based account as a Bitcoin wallet, leading to compatibility issues and preventing seamless integration with external wallet software.

Recommendation

We recommend updating the `SetWalletConfig` function to set the `CoinType` explicitly to `118`, the `CoinType` for Cosmos chains, or another unused number.

Status: Acknowledged

7. Incorrect `bech32` prefix in account keeper

Severity: Minor

In `app/keepers/keepers.go:112`, the account keeper is constructed with the `bech32prefix` parameter set to `sdk.Bech32PrefixAccAddr`, which is `"cosmos"`.

This is problematic because the correct `bech32` prefix should be `"hippo"`, as defined in `types/consensus/wallet.go:8`.

Recommendation

We recommend updating the `bech32` prefix to `"hippo"`.

Status: Resolved

8. CLI discards filters during genesis export

Severity: Informational

In `hippod/cmd/root.go`, the `AppExport` function allows users to specify a list of modules to be exported via the CLI.

However, in `app/export.go:17-44`, the `ExportAppStateAndValidators` function does not process the `modulesToExport` parameter and instead exports all modules unconditionally.

As a result, regardless of the CLI command's specified module list, the genesis state for all modules is always exported.

Recommendation

We recommend modifying `ExportAppStateAndValidators` to properly handle the `modulesToExport` parameter. This should involve filtering the exported modules based on the provided list, ensuring that only the intended modules are included in the output.

Status: Resolved

9. Misplaced configuration of initialization parameters

Severity: Informational

In `hippod/main.go:17-23`, `DefaultPowerReduction` is set to handle `DefaultHippoPrecision`, and `RegisterDenom` is executed for `DefaultHippoDenom`.

However, these operations would be more appropriately placed in the `init` function of `app.go` to ensure proper initialization and maintain code structure consistency.

Recommendation

We recommend moving `DefaultPowerReduction` handling and `RegisterDenom` execution to the `init` function in `app.go`.

Status: Resolved

10. Redundant definition of `MakeEncodingConfig` function

Severity: Informational

In `app/app.go:550-557`, the `makeEncodingConfig` function is defined, despite an existing implementation in `app/encoding.go`.

This duplication results in unnecessary redundancy and increases maintenance complexity.

Recommendation

We recommend removing the redundant `makeEncodingConfig` function from `app/app.go` and utilizing the existing implementation in `app/encoding.go` to ensure consistency and reduce code duplication.

Status: Resolved

11. Use 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:

- `app/inflation.go:31-32`

Additionally, the defined number differs from the one specified in the genesis file.

Recommendation

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

Status: Resolved