Audit Report

Sei Tendermint

v1.0

May 15, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Sei Labs Inc to perform a security audit of Sei Tendermint.

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 GitHub repository:

https://github.com/sei-protocol/sei-tendermint

Commit hash: `d86db708823ced38278adb89d124b37f3413bb4e`

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

Sei Tendermint is a forked version of Tendermint Core that is software for securely and consistently replicating an application on many machines. Tendermint Core performs Byzantine Fault Tolerant (BFT) State Machine Replication (SMR) for arbitrary deterministic, final state machines. The Sei team has customized several features, made improvements, and fixed some bugs from the upstream Tendermint Core.

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**, 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	Readability is generally in line with the upstream Tendermint Core repository.
Level of documentation	Medium	The CHANGELOG document is available in the repository. However, records of changes do not have direct links to their respective pull requests. Therefore, it is difficult to track and follow what was being discussed and the rationale behind design decisions.
Test coverage	Medium-High	Go test reports a 67.4% code coverage excluding automatically generated files.

Summary of Findings

No	Description	Severity	Status
1	Saving an invalid `LastResultsHash` and `AppHash` when hard rollback	Major	Resolved
2	Missing error handling for `setProposal` might cause unwanted state to be flushed to disk and invalid proposals to be added to the block	Minor	Resolved
3	Remove unused files and code	Informational	Resolved
4	Use of deprecated function	Informational	Resolved
5	Redundant use of span closing statements is inefficient	Informational	Resolved
6	Missing validation might lead to dangling pointer	Informational	Resolved
7	`TxNotifyThreshold` should be of type unsigned integer	Informational	Resolved

Detailed Findings

1. Saving an invalid `LastResultsHash` and `AppHash` when performing a hard rollback

Severity: Major

Hard rollback deletes the block and the app states to recover an incorrect application state transition. In `internal/state/rollback.go:117`, when `removeBlock` is true, the latest block is removed from the block store. However, in `internal/state/rollback.go:110-111`, `latestBlock` is used to store both `LastResultsHash` and `AppHash` in `rolledBackState` where it should use `rollbackBlock` when performing a hard rollback.

Recommendation

We recommend using `rollbackBlock` to store `LastResultsHash` and `AppHash` when hard rollback.

Status: Resolved

2. Missing error handling for `setProposal` might cause unwanted state to be flushed to disk and invalid proposals to be included in a block

Severity: Minor

In `internal/consensus/state.go:1037.go`, there is a call to method `setProposal` which returns an error. However, this error is not immediately handled, which might cause the proposal to be flushed to disk and created even when an error occurred.

One of the reasons why `setProposal` may return a non-nil error upon an invalid proposal signature as depicted in `internal/consensus/state.go:2333`. In that case, a proposal that has been signed by a wrong signature gets added to a block, even though it should have been discarded.

Recommendation

We recommend checking the error right after `setProposal` is called, and avoiding updating the state in case of an error.

Status: Resolved

3. Remove unused files and code

Severity: Informational

Across the codebase, various instances of unused files and code have been found.

Recommendation

The following are some recommendations to improve the overall code quality and readability

- The `evidence.pb.go` file in `proto/tendermint/types` package can be removed since the `evidence.proto` file no longer exists.
- Remove unused function `ABCIREsponseResultHash` in `state/store.go:8`.
- Remove unused function `CRandHex` in `crypto/random.go:19`.
- Remove unused function `SumTruncated` in `crypto/tmhash/hash.go:62`.
- Remove unused function `NewTruncated` in `crypto/tmhash/hash.go:55`.
- Remove unused function `Sha256` in `crypto/hash.go:7`.
- Remove unused function `MarshalIndent` in `libs/json/encoder.go:32`
- Remove unused type `sigs_verified` in `proto/tendermint/abci/types.proto:135`.

Status: Resolved

4. Use of deprecated function

Severity: Informational

The deprecated function `grpc.WithInsecure` is used in `abci/client/grpc_client.go:60`.

Recommendation

We recommend avoiding the use of deprecated functions.

Status: Resolved

5. Redundant use of span closing statements is inefficient

Severity: Informational

In the `finalizeCommit` method in `internal/consensus/state.go:2148`, the `storeBlockSpan` is closed. However, this operation is redundant as in line `2141` there is a deferred statement already closing the same span. Redundant code can make the code more complex or inefficient.

Recommendation

We recommend removing the close span statement in line `2148`, and relying on the defer logic to close spans, as done in other places of the codebase.

Status: Resolved

6. Missing validation might lead to dangling pointer

Severity: Informational

In the `OnStart` method in `internal/mempool/reactor.go:98`, the `channel` field from the `Reactor` type is not checked against the `nil` value. This can cause a dangling pointer, and the main thread to crash.

This is not a security concern in the current implementation, since upon node creation in `node/node.go:274` the mempool reactor is initialized and the channel is created by passing a callback `SetChannel` to method `AddChDescToBeAdded`.

Nevertheless, the fact that the mempool constructor initializes the channel as `nil`, and there is no further validation could potentially cause an issue in the future.

Recommendation

We recommend checking whether the pointer is `nil` or not, and only continuing execution if it is not.

Status: Resolved

7. `TxNotifyThreshold` should be of type unsigned integer

Severity: Informational

In `config/config.go:776`, `TxNotifyThreshold` is defined as an `int` but its value should never be negative. There is a check in the `ValidateBasic` function in the same file to ensure that this invariant is upheld. By using an unsigned integer type, this check would become redundant, which would be more efficient.

Recommendation

We recommend modifying the type from `int` to an unsigned integer, such as `uint32` or `uint64`.

Status: Resolved