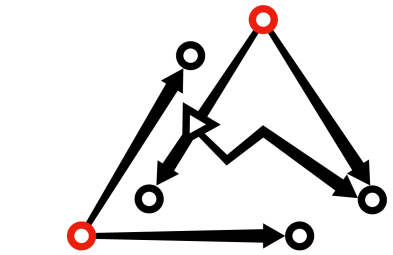

Least Authority

PRIVACY MATTERS

Bor Heimdall

Security Audit Report

Polygon

Initial Audit Report: 4 July 2023

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - System Design
    - Code Quality
    - Documentation
    - Scope
  - Specific Issues & Suggestions
    - Issue A: Unused and Incomplete Functions
  - Suggestions
    - Suggestion 1: Improve Test Suite
    - Suggestion 2: Check the Expression on Both Sides of the && Operator
    - Suggestion 3: Address TODO and FIXME Comments
    - Suggestion 4: Improve Error Handling, Limit and Avoid Using Panics
    - Suggestion 5: Update Documentation To Explain Design Decisions
    - Suggestion 6: Replace Introduced Panic With an Error
- About Least Authority
- Our Methodology

Overview

Background

Polygon has requested that Least Authority perform a security audit of their Bor and Heimdall implementations.

Project Dates

- **April 10 - May 22, 2023**: Initial Code Review *(Completed)*
- **May 26, 2023**: Delivery of Initial Audit Report *(Completed)*
- **June 10, 2023:** Verification Review *(Completed)*
- **July 4, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Alicia Blackett, Security Researcher and Engineer
- Steven Jung, Security Researcher and Engineer
- ElHassan Wanas, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Bor and Heimdall implementations followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Bor:
  https://github.com/maticnetwork/bor/pull/654
- Heimdall:
  https://github.com/maticnetwork/heimdall/pull/939

Specifically, we examined the Git revisions for our initial review:

- Bor: `e3582ca11bd44c5c85310ea1ac1bf0e4354e4457`
- Heimdall: `a5545a1df3b89a88b245908675dbf02cd5794346`

For the verification, we examined the Git revision:

- `02324f3b12489bc957d54a1c17cd5f75f6019832`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Bor:
  https://github.com/LeastAuthority/polygon-bor/pull/1
- Heimdall:
  https://github.com/LeastAuthority/polygon-heimdall/pull/1

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- RFC-106: https://www.notion.so/polygontechnology/RFC-106-6dce7364e469424890ea826dc122fae4
- Introduction to Polygon PoS: https://wiki.polygon.technology/docs/pos/polygon-architecture
- HF2_PIP_Draft.pdf *(shared with Least Authority via email on 15 February 2023)*

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of service (DoS) attacks and security exploits that would impact the implementation or disrupt its execution;
- Vulnerabilities within individual components and whether the interaction between the components is secure;
- Exposure of any critical information during interaction with any external libraries;
- Proper management of encryption and signing keys;
- Protection against malicious attacks and other methods of exploitation;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a security audit of developments in the Polygon's Bor and Heimdall implementations. Bor is a Go implementation of the Matic protocol (Polygon), which is a fork of Go Ethereum. It is a proof-of-stake (PoS) layer 2 designed to optimize transaction efficiency and provide quick transaction finality, relative to Ethereum. Heimdall is a validator node implementation that uses Tendermint Cosmos-SDK modules in a fork of Tendermint, called Peppermint, that provides compatibility with Ethereum.

The scope of this review consisted of modifications to the Bor and Heimdall codebases in the form of the pull requests (PR) referenced above. We focused our investigation on those changes, checking the areas of concern listed above to identify security vulnerabilities introduced by those changes.

In performing the audit, our team examined several areas for security vulnerabilities, including the access control implemented for security-critical functions, such as the signing of transactions and the validation of signatures.

To maintain synchrony between the Bor and Heimdall layers, a milestone snapshot (the root hash of the Merkle tree) of specific block heights is persisted. We looked at milestone edge cases, including block lengths greater than 12, in more than 100 previously stored milestones. Heimdall maintains synchrony with the Ethereum network by means of a similar mechanism with checkpoints representing the root hash of the Heimdall at certain block heights recorded on the Ethereum blockchain. Our team examined these mechanisms and did not identify any security vulnerabilities.

We attempted to bypass functions such as `validation` or `voting` before approving milestones. Although our team did not identify security vulnerabilities introduced by the pull requests, the issues and suggestions our team identified indicate that the implementation is in progress, with functionality yet to be implemented and missing code quality features, such as properly handled errors, comprehensive test coverage, in addition to a more comprehensive scope. As such, we recommend that the Polygon team conduct a comprehensive security audit of the Bor and Heimdall implementations once development is complete.

System Design

Our team found that security has generally been taken into consideration in the design and implementation of Bor and Heimdall as demonstrated by documented threat modeling to determine and mitigate or remediate potential attack vectors that could put the security of the system at risk. Our team also noted the implementation of recommended security practices, such as the use of separate signing keys and Heimdall validators.

Code Quality

Our team performed a manual review of the codebases and identified areas of improvement, including resolving TODOs and FIXMEs that diminish the ability to assess the intended functionality of the code and raise questions about the completeness of the implementation (Suggestion 3). We also found that error handling in the implementation can be improved by limiting and avoiding the use of panics (Suggestion 4). In addition, our team found instances of empty functions and unused code that could affect the intended behavior of the implementation (Issue A).

Tests

Our team found that the functionality introduced by both pull requests is not sufficiently tested. Although the repositories include tests that are implemented in the fork, as a result of the modifications, some of those tests no longer work. We recommend that comprehensive security and functional tests be implemented and that the failing tests be fixed (Suggestion 1).

Documentation

The project documentation provided for this review was sufficient and offered a detailed description of the general architecture of the Polygon ecosystem and how the Bor and Heimdall implementations interact with each other. However, the HF2_pip_draft document was out of date, and the RFC-106 document was more helpful. Although the documentation describes the system clearly, we found that it can be improved by including design decisions and rationalizations (Suggestion 5).

Code Comments

Both codebases include code comments that sufficiently describe the intended behavior of security-critical functions and components.

Scope

The Bor and Heimdall implementations compose two components of the Polygon PoS network. The scope of this review consisted of two pull requests within those component repositories. The in-scope code for this review consisted of the modifications to the codebases made in those PRs, with the original branches being out of scope. Hence, in order to contextualize the PRs, our team had to refer to the out-of-scope code. For future audits, we recommend that the entire repositories of both Bor and Heimdall be in scope, preferably as separate audits.

Dependencies

While the main focus of the audit was the functionality of the milestone, there were concerns about some issues, as documented above, which were found in other areas of the repository.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Unused and Incomplete Functions	Resolved
Suggestion 1: Improve Test Suite	Unresolved
Suggestion 2: Check the Expression on Both Sides of the && Operator	Resolved
Suggestion 3: Address TODO and FIXME Comments	Partially Resolved
Suggestion 4: Improve Error Handling, Limit and Avoid Using Panics	Unresolved
Suggestion 5: Update Documentation To Explain Design Decisions	Unresolved
Suggestion 6: Replace Introduced Panic With an Error	Resolved

Issue A: Unused and Incomplete Functions

Location

`polygon-bor/les/api_backend.go#L362`

`polygon-bor/les/api_backend.go#L369`

Synopsis

Our team found instances of functions that are defined but not implemented yet. This makes the intended functionality unclear and inhibits code review.

Mitigation

We recommend removing unused code from the implementations.

Status

The Polygon team has added the missing implementations for purging the whitelisted checkpoint and the milestone.

Verification

Resolved.

Suggestions

Suggestion 1: Improve Test Suite

Synopsis

The test suites in the repositories do not test all functionality, with coverage ranging from 6% under Consensus/Bor to 95% for the already implemented Ethereum code in the Bor repository. In addition, there are failing tests in the Bor repository: `TestServer_DeveloperMode` and `TestHashMemoryExhaustionAttack`.

There are also a number of failing tests in the Heimdall repository: `TestParseSubmitProposalFlags`, `TestValidateBasic`, and `TestNewValidator`. The coverage in the Heimdall repository ranges from 0% for the `slashing` module to 87% for the `auth` module. The failing tests are related to functionality introduced in the Heimdall.

Mitigation

We recommend that the Polygon team expand the test suite to cover all new functionality introduced by Polygon, and fix the failing tests.

Status

At the time of the verification, the suggested mitigation has not been resolved.

Verification

Unresolved.

Suggestion 2: Check the Expression on Both Sides of the && Operator

Location

[polygon-bor/tests/bor/bor_milestone_test.go#L477]

Synopsis

There is an identical comparison expression on both sides of the && operator.

Mitigation

We recommend checking if the expression should be changed to `blockHeaderVal0.Number.Uint64() > 12 && blockHeaderVal1.Number.Uint64() > 12`.

Status

The Polygon team has implemented the fix for the test case.

Verification

Resolved.

Suggestion 3: Address TODO and FIXME Comments

Location

polygon-bor/eth/backend.go#L798

polygon-bor/eth/downloader/whitelist/finality.go#L48

polygon-bor/eth/downloader/whitelist/service.go#L215

polygon-bor/eth/downloader/whitelist/milestone.go#L133

polygon-bor/eth/downloader/whitelist/milestone.go#L160

polygon-bor/eth/downloader/whitelist/milestone.go#L234

Synopsis

There are some TODO and FIXME comments in the codebase, which may lead to a lack of clarity and cause confusion about the completeness of the implementation. Resolving such comments prior to a comprehensive security audit of the code allows security researchers to better understand the full intended functionality of the code, indicates completion, and increases readability and comprehension.

Mitigation

We recommend that the comments either be resolved or removed from the codebases.

Status

The Polygon team has removed all the FIXME and TODO comments, except for those in polygon-bor/eth/downloader/whitelist/milestone.go#L234.

Verification

Partially Resolved.

Suggestion 4: Improve Error Handling, Limit and Avoid Using Panics

Location

Examples (non-exhaustive):

ethclient/signer.go#L58

ethclient/signer.go#L61

eth/downloader/peer.go#L85

eth/downloader/peer.go#L88

core/types/transaction_signing.go#L119

Synopsis

There are multiple instances in the code that would trigger a panic in case of an error. Functions that can cause the code to panic at run time may lead to denial of service. There are also instances in the code where errors are not handled at all.

Mitigation

We recommend refactoring the code and removing panics where possible. One of the possible improvements is to propagate errors to the caller and handle them on the upper layers. Note that error handling does not exclude using panics. In addition, if a caller can return an error, the callee function may not panic but, instead, propagate an error to the caller.

Status

Our team found that panics are still in place. Hence, at the time of the verification, the suggested mitigation has not been resolved.

Verification

Unresolved.

Suggestion 5: Update Documentation To Explain Design Decisions

Synopsis

The technical documentation is comprehensive in many regards, as it accurately explains the ways in which the system works. However, our team found that the decisions that led to the current design are not always documented. For example, it was not immediately clear why the Heimdall system, which already is a layer 2 to Ethereum, needed Bor as a layer 3 system.

These documents need not necessarily be facing people using Polygon, but people developing and reviewing it.

Mitigation

We recommend updating project documentation to explain the rationale behind design decisions.

Status

At the time of verification, our team found no evidence that the project documentation has been improved.

Verification

Unresolved.

Suggestion 6: Replace Introduced Panic With an Error

Location

`consensus/bor/heimdall/span/spanner.go#L119`

Synopsis

The usage of `panic` in Go is not a recommended practice. The PR for Bor introduces a change, which replaces a regular `error` return with a `panic`.

Mitigation

We recommend keeping the `error` return as is and handling it appropriately in the `caller`.

Status

The Polygon team has implemented the remediation as recommended, and the implementation now returns an `error` instead of emitting a `panic`.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.