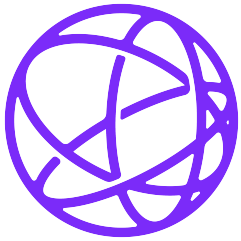


Celestia

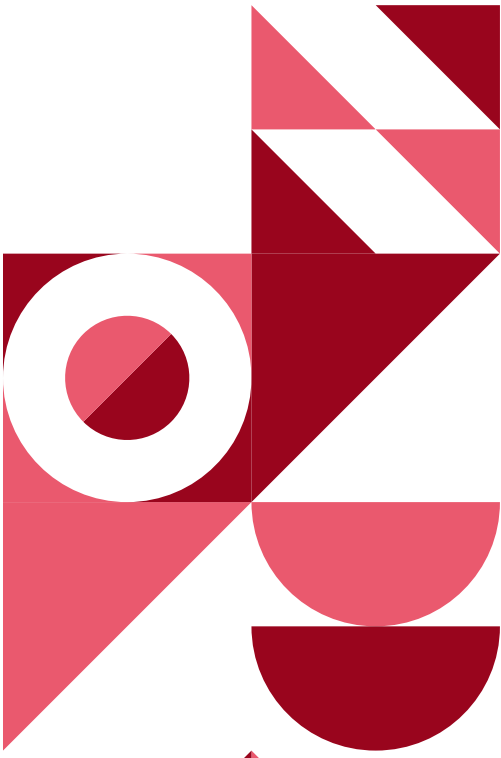
Audit

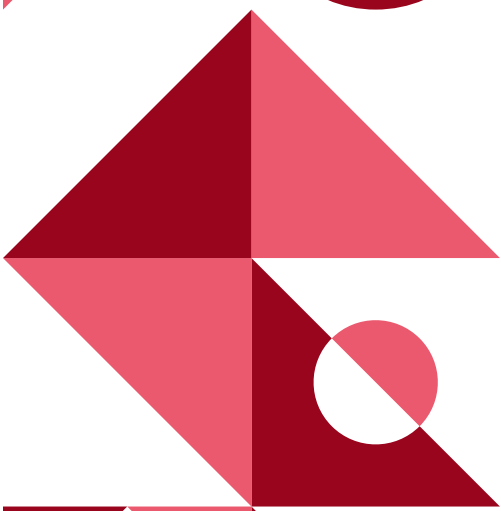
Presented by:

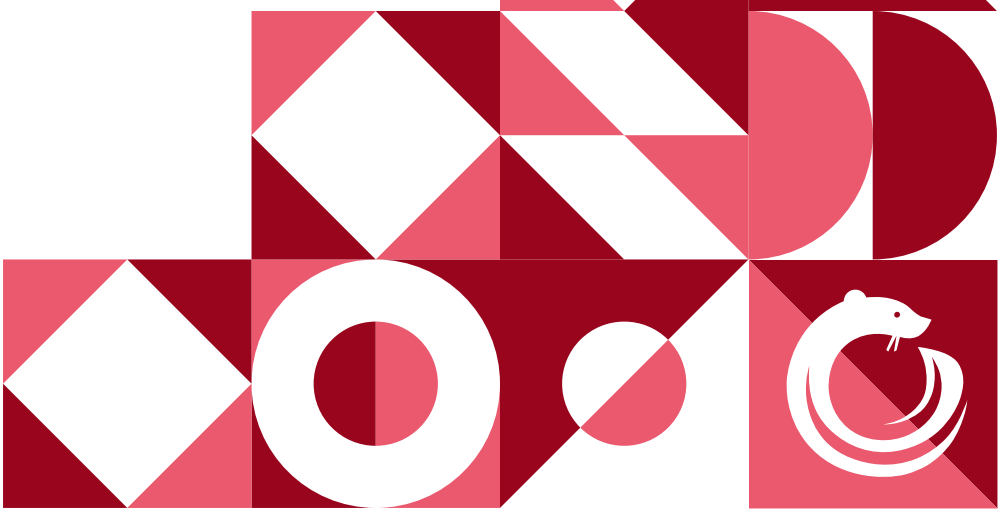
OtterSec contact@osec.io

James james.wang@osec.io

Robert Chen r@osec.io







Contents

Appendices

01 | Executive Summary

Overview

Celestia engaged OtterSec to assess the `op-stack` program. This assessment was conducted between December 1st and December 12th, 2023. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 3 findings throughout this audit engagement.

In particular, we discovered a medium-risk vulnerability regarding the absence of a connection timeout in the Celestia remote procedure call client, which may hang Optimism indefinitely (OS-OPS-ADV-00).

We also made recommendations around Optimism's state derivation process, which lacks a fallback mechanism while retrieving data from Celestia (OS-OPS-SUG-00). Additionally, we advised specific optimizations for the configuration handling process by moving Celestia remote procedure call configuration fetching into unified configuration parsing files and removing default values (OS-OPS-SUG-01).

02 | Scope

The source code was delivered to us in a git repository at github.com/celestiaorg/optimism/commits/celestia-develop. This audit was performed against commits between 6ce4e61 and f88e4a2.

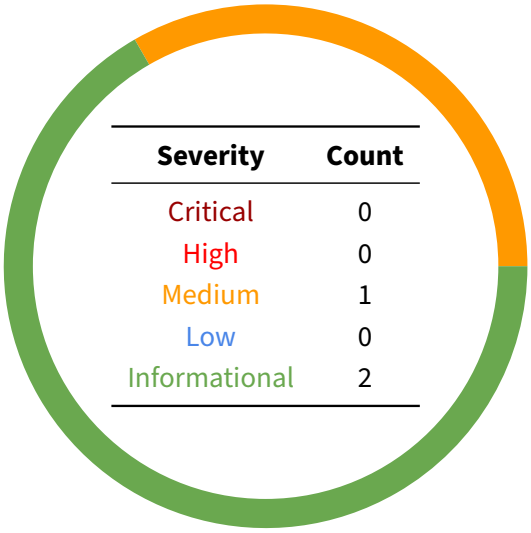
A brief description of the programs is as follows:

Name	Description
`op-stack`	Celestia integration within the OP Stack allows users to settle data on the less costly Celestia chain instead of the original L1.

03 | Findings

Overall, we reported 3 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	1
Low	0
Informational	2

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-OPS-ADV-00	Medium	Resolved	The absence of a connection timeout in the Celestia remote procedure call client may hang Optimism indefinitely.

OS-OPS-ADV-00 [med]| Inability To Timeout Connections

Description

The current remote procedure call client (`DAClient`) used to connect to the Celestia network, lacks a timeout configuration during establishing the connection. This omission may result in liveliness issues within the Optimism L2 chain if Celestia's RPC servers accept connections but fail to respond to calls.

op-node/rollup/da_client.go RUST

```
type DAClient struct {
        Client *proxy.Client
}

func NewDAClient(rpc string) (*DAClient, error) {
        client := proxy.NewClient()
        err := client.Start(rpc,
            ↪  grpc.WithTransportCredentials(insecure.NewCredentials()))
        if err != nil {
                return nil, err
        }
        return &DAClient{
                Client: client,
        }, nil
}
```

Within `da_client`, the `NewDAClient` function utilizes `client.Start` to initiate a connection to the Celestia remote procedure call server. However, the absence of a timeout configuration in this process may result in a blocking scenario, particularly when posting transactions to Celestia for storage. If the server is experiencing problems. indefinite blockage may occur, forcing the entire system to rely on the `DAClient` to hang indefinitely while awaiting the response to the remote procedure call. This, in turn, may impact the liveliness of the Optimism system.

Remediation

Add a timeout mechanism for Celestia gRPC to properly fall back to Layer 1 Ethereum if Celestia is unresponsive or encounters issues.

Patch

Resolved in 9f21d29.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-OPS-SUG-00	Optimism's state derivation process lacks graceful fallback mechanisms for retrieving data from Celestia.
OS-OPS-SUG-01	The configuration handling process may be optimized.

OS-OPS-SUG-00 | Handling Get Failures

Description

There is a potential limitation in Optimism's system when interacting with Celestia for data retrieval. Optimism interacts with Celestia in two scenarios:

1. **Posting Data for Storage**: This involves sending data to Celestia for storage.
2. **Getting Data for State Derivation**: This involves retrieving data from Celestia to derive the state.

op-node/rollup/derive/l1_retrieval.go RUST

```
// NextData does an action in the L1 Retrieval stage
// If there is data, it pushes it to the next stage.
// If there is no more data open ourselves if we are closed or close ourselves if
    ↪ we are open
func (l1r *L1Retrieval) NextData(ctx context.Context) ([]byte, error) {
        [...]
        data, err := l1r.datas.Next(ctx)
        if err == io.EOF {
                l1r.datas = nil
                return nil, io.EOF
        } else if err != nil {
                // CalldataSource appropriately wraps the error so avoid double
                    ↪ wrapping errors here.
                return nil, err
        } else {
                return data, nil
        }
}
```

While failures in posting data can be managed gracefully by switching to Ethereum Layer 1 when Celestia encounters issues, the situation becomes more critical when it comes to retrieving data from Celestia. Currently, there is no mechanism in place to handle this scenario gracefully.

The inability to retrieve data from Celestia for state derivation means that the derivation process may become impossible during Celestia's remote procedure call failures. This limitation weakens the Optimism invariant where Layer 2 must always be derivable from Etheruem Layer 1, introducing a potential point of failure in the derivation process.

Remediation

While this limitation is inevitable for solutions that settle data outside of the Ethereum chain, we believe it should be documented and made clear to developers for future consideration in system design and operation.

OS-OPS-SUG-01 | Centralized Configuration Handling

Description

Rather than retrieving the Celestia remote procedure call configuration on demand, as currently implemented, it may be more beneficial to pre-fetch the configuration within the unified configuration parsing files. Collecting configuration fetching and parsing may increase the maintainability and readability of the code.

op-batcher/batcher/driver.go RUST

```
func (l *BatchSubmitter) Start() error {
        [...]
        daRpc := os.Getenv("OP_BATCHER_DA_RPC")
        if daRpc == "" {
                daRpc = "localhost:26650"
        }
        daClient, err := rollup.NewDAClient(daRpc)
        if err != nil {
                return err
        }
        [...]
}
```

Additionally, rather than falling back to default values when certain configurations are missing, an error should be thrown, serving as an assertive mechanism to remind users to set the configuration parameters. This ensures that critical settings are explicitly provided and prevents the program from running with potentially incorrect values.

Remediation

Move the Celestia remote procedure call configuration fetching process into the unified configuration parsing files and ensure the program panics if users fail to set the config parameters.

Patch

The suggestions are partially adopted in f1b4c28. After consideration, the Celestia team decided to retain a default `daRpc` value for integration ease.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings section.

Critical	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
High	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
Medium	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
Low	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.