Zellic

October 15, 2024

Prepared for
Jordan Oroshiba
Astria

Prepared by
Ayaz Mammadov
Nan Wang
Avraham Weinstock
Zellic

Astria Shared Sequencer
Blockchain Security Assessment

Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Astria from September 30th to October 10th, 2024. During this engagement, Zellic reviewed Astria Shared Sequencer's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is data written to and retrieved from Celestia correctly?
- Are fee calculations correct?
- Does the Conductor properly make progress on restarts?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

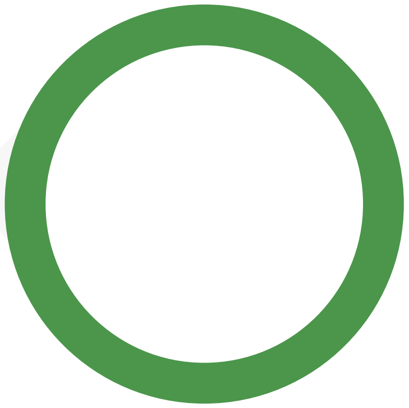
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Astria Shared Sequencer crates, we discovered three findings, all of which were low impact.

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	3
⬜ Informational	0



2. Introduction

2.1. About Astria Shared Sequencer

Astria contributed the following description of Astria Shared Sequencer:

> Astria is a Shared Sequencer Network with Celestia underneath. It provides ordering guarantees with soft commitments to chains, relying on Celestia's DAS network to provide firm commitments and broad data availability.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the crates.

Nondeterminism. Nondeterminism is a leading class of security issues on Cosmos. It can lead to consensus failure and blockchain halts. This includes but is not limited to vectors like wall-clock times, map iteration, and other sources of undefined behavior (UB) in Go.

Arithmetic issues. This includes but is not limited to integer overflows and underflows, floating-point associativity issues, loss of precision, and unfavorable integer rounding.

Complex integration risks. Several high-profile exploits have been the result of unintended consequences when interacting with the broader ecosystem, such as via IBC (Inter-Blockchain Communication Protocol). Zellic will review the project's potential external interactions and summarize the associated risks. If applicable, we will also examine any IBC interactions against the ICS Specification Standard to look for inconsistencies, flaws, and vulnerabilities.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Astria Shared Sequencer Crates

Type	Rust
Platform	Cosmos

Target	astria
Repository	https://github.com/astriaorg/astria ↗
Version	`8e3a474cc41cc655892b9d2ee9d8438ad6da27b2`
Programs	`astria-conductor` `astria-sequencer-relayer`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.7 person-weeks. The assessment was conducted by three consultants over the course of two calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Ayaz Mammadov
Engineer
ayaz@zellic.io ↗

Nan Wang
Engineer
nan@zellic.io ↗

Avraham Weinstock
Engineer
avi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 30, 2024	Start of primary review period
October 1, 2024	Kick-off call
October 10, 2024	End of primary review period

3. Detailed Findings

3.1. Log parsing issue in `extract_required_fee_from_log`

Target	astria-sequencer-relayer/src/relayer/celestia_client/mod.rs		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The `extract_required_fee_from_log` function currently makes a best-effort attempt to parse the log returned as part of Celestia's response to a Cosmos transaction when a transaction fails due to insufficient fees. However, it does not handle parsing failures, which can result in a fallback to the original fee-calculation logic without any adjustment. This can lead to repeated failures in fee-submission attempts, creating a potential DOS issue.

Impact

If parsing the log fails, the function falls back to the default logic in `calculate_fee`. This can cause an infinite retry loop where the transaction is repeatedly submitted with an insufficient fee, leading to a system hang or a DOS scenario.

```
/// `log`'s value for this case currently looks like:
/// "insufficient fees; got: 1234utia required: 7980utia: insufficient fee"
/// We'll make a best-effort attempt to parse, but this is just a failsafe to
    check the
/// new calculated fee using updated Celestia costs is sufficient, so if
    parsing fails
/// we'll just log the error and otherwise ignore.
fn extract_required_fee_from_log(celestia_broadcast_tx_error_log: &str) ->
    Option<u64> {
    const SUFFIX: &str = "utia: insufficient fee";
    // Should be left with e.g. "insufficient fees; got: 1234utia required:
    7980".
    let Some(log_without_suffix)
    = celestia_broadcast_tx_error_log.strip_suffix(SUFFIX) else {
        warn!(
            celestia_broadcast_tx_error_log,
            "insufficient gas error doesn't end with '{SUFFIX}'"
        );
        return None;
    };
    // Should be left with e.g. "7980".
    let Some(required) = log_without_suffix.rsplit(' ').next() else {
        warn!(
            celestia_broadcast_tx_error_log,
            "insufficient gas error doesn't have a space before the required
    amount"
        );
        return None;
    };
    match required.parse::<u64>() {
        Ok(required_fee) => {
            info!(
                required_fee,
                "extracted required fee from broadcast transaction response raw
    log"
            );
            Some(required_fee)
        }
        Err(error) => {
            warn!(
                celestia_broadcast_tx_error_log, %error,
                "insufficient gas error required amount cannot be parsed as
    u64"
            );
            None
        }
    }
}
```

Recommendations

We recommend the following:

- Improve the robustness of the log parsing in the `extract_required_fee_from_log` function, such as logging detailed errors and possibly adjusting the fee by a reasonable percentage to avoid repeated failures.
- Implement more comprehensive tests for fee calculation, including nightly tests to ensure the log-parsing and fee-submission mechanisms are functioning as expected.

Remediation

TBD

3.2. Soft-block channel capacity misconfiguration

Target	astria-conductor/src/executor/mod.rs		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The `soft_blocks` channel is used for communicating soft blocks (which have been processed by the Sequencer but not Celestia, as opposed to firm blocks, which have also been processed by Celestia).

Its capacity is tied to the `celestia_block_variance` value in the rollup node's genesis configuration. If this value is set to zero, soft blocks will never be sent or executed.

Impact

In the Executor's init method, the soft block channel's capacity is set based on `calculate_max_spread`, which multiplies `celestia_block_variance` by six:

```
fn calculate_max_spread(&self) -> usize {
    usize::try_from(self.state.celestia_block_variance())
        .expect("converting a u32 to usize should work on any architecture
    conductor runs on")
        .saturating_mul(6)
}
```

If `celestia_block_variance` is zero, it results in a `max_spread` of zero, preventing soft blocks from being processed:

```
if let Some(channel) = self.soft_blocks.as_mut() {
    channel.set_capacity(max_spread);
}
```

This creates a situation where soft blocks are never sent or executed. Notably, in soft-only mode, `celestia_block_variance` and the soft-block channel capacity do not appear to have any direct functional relationship, yet a zero variance still prevents execution.

Recommendations

Ensure a minimum value for `celestia_block_variance` or set a default value for soft-only mode to decouple the logic and guarantee that soft blocks can be processed in any mode.

Remediation

TBD

3.3. Potential overflow in `celestia_block_variance` to `usize` conversion

Target	astria-conductor/src/executor/mod.rs		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

A potential overflow occurs when converting a `u64` to `usize` for the `celestia_block_variance`.

Impact

The issue arises from trying to convert `celestia_block_variance` (which is a `u64`) to `usize`. On 32-bit systems, this conversion could lead to a panic if `celestia_block_variance` exceeds `u32::MAX`. Although unlikely, this could result in a DOS if the system encounters a value larger than `u32` during the conversion.

Currently, Conductor only compiles on 64-bit architectures due to restrictions in astria-core, where a compile-time assertion prevents 32-bit system compatibility. The `calculate_max_spread` function still contains an incorrect expect message regarding the conversion, which could lead to confusion.

```
/// Calculates the maximum allowed spread between firm and soft commitments
    heights.
///
/// The maximum allowed spread is taken as `max_spread = variance * 6`, where
    `variance`
/// is the `celestia_block_variance` as defined in the rollup node's genesis
    that this
/// executor/conductor talks to.
///
/// The heuristic 6 is the largest number of Sequencer heights that will be
    found at
/// one Celestia height.
///
/// # Panics
/// Panics if the `u32` underlying the celestia block variance tracked in the
    state could
/// not be converted to a `usize`. This should never happen on any reasonable
    architecture
/// that Conductor will run on.
```

```
fn calculate_max_spread(&self) -> usize {
    usize::try_from(self.state.celestia_block_variance())
        .expect("converting a u32 to usize should work on any architecture
    conductor runs on")
        .saturating_mul(6)
}

pub struct GenesisInfo {
    /// The rollup id which is used to identify the rollup txs.
    rollup_id: RollupId,
    /// The Sequencer block height which contains the first block of the rollup.
    sequencer_genesis_block_height: tendermint::block::Height,
    /// The allowed variance in the block height of celestia when looking for
    sequencer blocks.
    celestia_block_variance: u64,
}
```

Recommendations

We recommend the following:

- Update the expect message to correctly describe the `u64`-to-`usize` conversion.
- Add a compile-time check within Conductor itself to ensure it only runs on 64-bit systems.

Remediation

TBD

4. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

4.1. Component: Conductor

The Conductor's main responsibility is to connect the shared Sequencer to the execution layer, where the execution layer can be any node that executes transactions and returns an execution hash (for example, Geth).

The conductor receives blocks in two ways — either from the shared Sequencer directly or from the data-availability layer (i.e., Celestia). Blocks received from the shared sequencer are filtered for transactions that match the Conductor's rollup chain and then sent to the execution layer for execution.

Transactions received from the Sequencer are marked as "soft" as soon as they are executed. These transactions are not considered finalized until the same block is also fetched from Celestia, which means that Celestia acts as the ultimate source of truth in this scenario.

The three subcomponents of the Conductor are as follows:

1. The Executor
2. The Sequencer Reader
3. The Celestia Reader

Since the last audit, the following changes have been made to the Conductor:

1. Data validation. Previously, corrupted data retrieved from Celestia would cause the system to crash. Now, after fetching data from Celestia, Conductor performs filtering and metadata verification to ensure the data's format and keys are correct, guaranteeing its usability for rollup playback.

2. Automatic restart. Conductor now supports automatic restarts, preventing crashes due to timing issues.

3. Bridge-deposit format update. The format for bridge deposits has been updated to prevent transaction hash collisions.

Subcomponent: Executor

The Executor essentially does two things in parallel:

1. It receives soft blocks from the sequencer and executes the transactions in them immediately, and it ensures that the block height matches the expected height. These executed blocks are then inserted into a list to await finalization.
2. It receives firm blocks from the data-availability layer (i.e., Celestia). These blocks are only executed if a corresponding soft block has not already been executed. Firm blocks are considered finalized.

For execution, the block is sent to the rollup execution layer node, which can be basically anything as long as it returns an execution hash. For example, a node like Geth or Erigon can be used for EVM-compatible transactions.

After execution, it updates the commitment state stored in the rollup execution layer node via gRPC, ensuring that the Conductor can resume from the correct state after a restart.

Subcomponent: Sequencer Reader

The Sequencer Reader component uses a Sequencer node's gRPC to fetch sequencer blocks. These blocks are sent through to the Executor marked as soft. If the executor channel is full, the blocks are scheduled and handled later.

Additionally, if the execution of soft blocks is significantly faster than firm blocks, the process may also temporarily halt.

Subcomponent: Celestia Reader

The Celestia Reader component fetches blobs containing sequencer block headers and rollup transactions from Celestia via JSON-RPC, validates them, and forwards them as firm blocks to the Executor.

The Celestia Reader's `FetchConvertVerifyAndReconstruct` task does the following:

- Fetches the block-header blobs and rollup blobs from Celestia, with exponential backoff on retries
- Attempts to decode the blobs to their expected types, dropping blobs that fail to decompress or decode
- Verifies, while decoding `SubmittedMetadata`, a Merkle proof that the rollup transaction root is included in the block
- Discards headers for blocks that are older than the current firm height
- Checks that each block header's chain ID and block hash agree with the sequencer's block for that height (fetched and cached from sequencer) and that the block was signed by a two-thirds majority of the sequencer's validators
- Reconstructs blocks from the headers and rollups blobs, dropping rollup blobs that do not have a valid inclusion proof for their associated header, which ensures that all and only the transactions associated with this sequencer block are included in the reconstructed block

- Yields the reconstructed blocks to the Celestia Reader's main loop, which inserts them into its block cache

The Celestia Reader's main loop acts as a scheduler:

- It schedules `FetchConvertVerifyAndReconstruct` tasks when Celestia's height is ahead of what has been processed and there are not too many reconstruction tasks pending.
- It handles shutdown requests from elsewhere in the conductor.
- It tracks which Celestia height the executor has last accepted.
- If it has reconstructed the block for the current firm height, it submits it as a firm block to the executor.
- It inserts reconstructed blocks into its block cache.
- It polls Celestia for its latest height.

4.2. Component: Sequencer relayer

The Sequencer relayer's main responsibility is to fetch blocks from the Sequencer and submit them to Celestia.

It contains a couple of subcomponents:

1. The API server
2. The relayer

Since the last audit, the following changes have been made to the relayer:

1. Data submission. The relayer has moved from using a Celestia node as an intermediary to communicating directly with the consensus node. This simplifies the data-handling process, as the system now only requires reading from a single file containing the key. Given this change, it is important to securely manage the key file.

2. State checking. By interacting directly with the Celestia application, the relayer now supports polling to check the submission status of data, ensuring transactions are confirmed as included in Celestia. This improves management of in-flight data.

3. Lock file. The lock-file logic has been updated to handle restart situations more effectively, preventing repeated or missed data submissions. Two separate lock files have been merged into a single, simplified lock file.

Subcomponent: API Server

The API server has three routes:

1. `/healthz` — This returns whether the relayer state is healthy, which is true when the relayer is connected to both the Sequencer and Celestia.
2. `/readyz` — This returns whether the relayer state is ready, which is true if the relayer currently has a block from the sequencer and a data-availability height from Celestia.
3. `/status` — This returns the `relayer_state` as a JSON object.

The `relayer_state` object looks as follows:

```
pub(crate) struct StateSnapshot {
    ready: bool,

    celestia_connected: bool,
    sequencer_connected: bool,

    latest_confirmed_celestia_height: Option<u64>,

    latest_fetched_sequencer_height: Option<u64>,
    latest_observed_sequencer_height: Option<u64>,
    latest_requested_sequencer_height: Option<u64>,
}
```

Subcomponent: Relayer

The relayer's main responsibility is to fetch blocks from the Sequencer using gRPC and submit them to Celestia.

The submitter task is sent blocks asynchronously as they are fetched from the Sequencer. It converts `SequencerBlock` objects to `Blob`s that can be submitted to Celestia. This `Blob` structure is as follows:

```
pub struct Blob {
    /// A [`Namespace`] the [`Blob`] belongs to.
    pub namespace: Namespace,
    /// Data stored within the [`Blob`].
    #[serde(with
    = "celestia_tendermint_proto::serializers::bytes::base64string")]
    pub data: Vec<u8>,
    /// Version indicating the format in which [`Share`]s should be created
    from this [`Blob`].
    ///
    /// [`Share`]: crate::share::Share
    pub share_version: u8,
    /// A [`Commitment`] computed from the [`Blob`]s data.
    pub commitment: Commitment,
  }
```

The converted `Blobs` are then submitted to Celestia via gRPC. It automatically retries submission for a while in case of unexpected failures, but ultimately it will quit after `u32::MAX` tries. It polls Celestia to ensure that the transaction it submitted was included.

5. Assessment Results

At the time of our assessment, the reviewed code for the Sequencer relayer was deployed to Astria's testnet, and the reviewed code for the Conductor was not deployed.

During our assessment on the scoped Astria Shared Sequencer crates, we discovered three findings, all of which were low impact.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.