Zellic

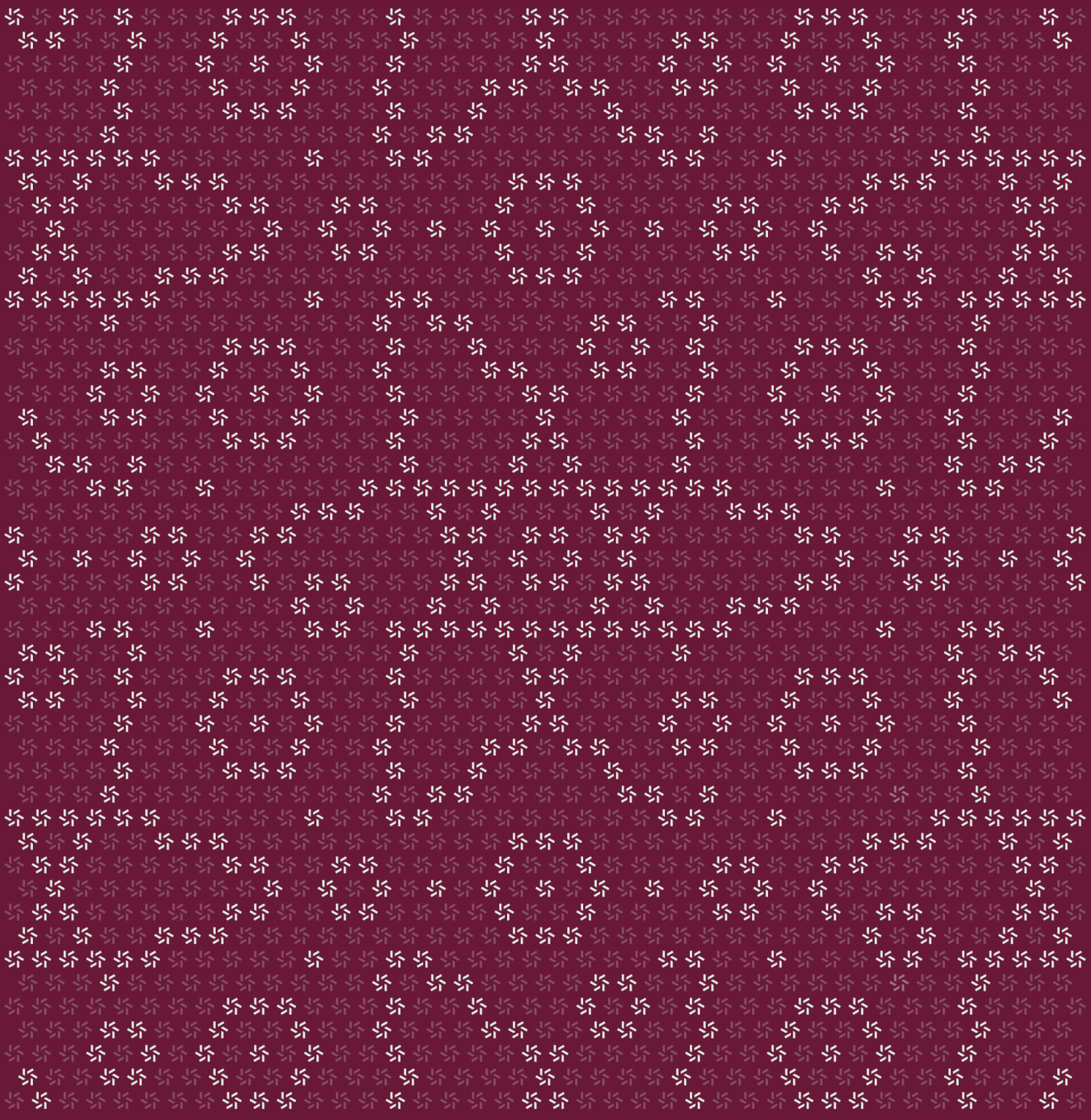
Prepared for
Jordan Oroshiba
Astria

Prepared by
Jaeeu Kim
Jisub Kim
Ayaz Mammadov
Zellic

August 3, 2024

Astria Bridge

Smart Contract 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 July 15th to July 31st, 2024. During this engagement, Zellic reviewed Astria Bridge'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:

- Could withdrawals fail to be processed by the off-chain withdrawer service after execution in the rollup-side smart contract, resulting in loss of user funds?
- Could the withdrawer service cause double spends by including the same withdrawal twice or processing the same block twice under different nonces?
- Could IBC failures/time-outs be improperly handled, leading to loss of user funds or double spending through faulty refunds?
- Could a malicious user drain funds from the bridge?
- Could a malicious user break the Astria bridge service?
- Could a malicious user execute unexpected behavior on the Astria bridge?

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
- IBC replayer

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 Bridge modules, we discovered five findings. No critical issues were found. One finding was of high impact, three were of medium impact, and one was of

low impact.

Additionally, Zellic recorded its notes and observations from the assessment for Astria's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	1
Medium	3
Low	1
Informational	0

2. Introduction

2.1. About Astria Bridge

Astria contributed the following description of Astria Bridge:

> The Astria sequencing layer provides a native bridging protocol which allows users bridging assets from the sequencing layer (and thus using IBC) into and out of rollups.
>
> The Astria Rollup Bridge provides functionality for bridging any tokens supported by the Astria sequencing layer into rollups built on top of it.
>
> The design is similar to existing Ethereum L1-to-rollup bridges in that funds are locked on the sequencing layer and a synthetic deposit is derived on the rollup.
>
> Burning the rollup-side funds will then cause the trusted bridge withdrawer service to issue a withdrawal transaction on the sequencing layer—either to a sequencing layer address or to a different chain using ICS20 withdrawals.

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:

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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped modules itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Astria Bridge Modules

Type	Rust
Platform	Cosmos
Target	astria-bridge-contracts
Repository	https://github.com/astriaorg/astria-bridge-contracts ↗
Version	`4580ffc0747f463e304214bb29848e21e4e93e32`
Programs	`src/AstriaBridgeableERC20.sol` `src/AstriaWithdrawer.sol`
Target	astria
Repository	https://github.com/astriaorg/astria ↗
Version	`bb2f96c01607a30806cb2195b6a7feb9ca325826`

Programs	`crates/astria-bridge-contracts/src/lib.rs`
	`crates/astria-bridge-withdrawer/*`
	`crates/astria-cli/src/cli/bridge.rs`
	`crates/astria-cli/src/commands/bridge/*`
	`tools/solidity-compiler/src/*`

2.4. Project Overview

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

Contact Information

The following project manager was associated with the engagement:

Jacob Goreski
Jr. Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Jaeeu Kim
Engineer
jaeeu@zellic.io

Jisub Kim
Engineer
jisub@zellic.io

Ayaz Mammadov
Engineer
ayaz@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

July 15, 2024	Kick-off call
July 15, 2024	Start of primary review period
July 31, 2024	End of primary review period

3. Detailed Findings

3.1. Invalid address could break down the bridge withdrawer

Target	astria-bridge-withdrawer		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

The watcher of the bridge withdrawer is responsible for monitoring the bridge contract's event log. When a user sends a transaction to the bridge address with parameters in rollup, the bridge withdrawer will listen for this event to submit a withdraw action to the sequencer.

However, in watcher of the bridge withdrawer, the destination chain address is not validated during logging. If the destination chain address is invalid, the bridge withdrawer crashes during parsing the destination address as `bech32m` format.

```
// ...
function withdrawToSequencer(string calldata destinationChainAddress)
    external payable sufficientValue(msg.value) {
    emit SequencerWithdrawal(msg.sender, msg.value, destinationChainAddress);
}
// ...
```

Impact

A malicious user could send a transaction to the bridge withdrawer with an invalid destination chain address. This would cause the bridge withdrawer to crash, preventing it from processing any further transactions.

Below is a proof of concept for this issue:

```
cast send 0xa58639fb5458e65e4fa917ff951c390292c24a15 --private-key
    $PK 'withdrawToSequencer(string)' "bbbb" -r
    http://executor.astria.localdev.me --value 100000000000000000
```

```
2024-07-19T13:33:22.398641Z ERROR astria_bridge_withdrawer::bridge_withdrawer:
    task returned with error task="ethereum watcher" error={"0": "block
    handler exited", "1": "failed to sync from next rollup block height", "2":
    "failed to get and send events at block", "3": "failed getting actions for
    block; block hash: `0xd3d1…54cf`, block height: `230`", "4": "failed to
```

```
    parse destination chain address as Astria address for a bridge unlock",
    "5": "failed decoding provided bech32m string", "6": "parsing failed",
    "7": "character error", "8": "invalid character (code=b)"}
// ...
2024-07-19T13:33:22.399185Z INFO astria_bridge_withdrawer: withdrawer stopped
```

Recommendations

Consider checking the destination chain address before logging the event.

Remediation

This was remediated in commit 3e24c50fc1daec666a51e93756268512fb098182 ↗ by changing the function `get_and_forward_block_events` to not error if the action fetcher returns errors instead the failed actions are filtered out and dropped.

3.2. Arbitrary withdrawal could be executed by bridge admin

Target	astria-bridge-withdrawer		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

In astria-bridge-withdrawer, `bridge address` and `withdrawer address` have permission to withdraw assets from the bridge.

For protocol design, astria-bridge-withdrawer is responsible for monitoring the bridge contract's event log from rollup and submitting the withdrawal action to the sequencer using private keys of the `bridge address` and `withdrawer address`.

However, `bridge address` and `withdrawer address` could execute a transaction directly in the sequencer without a transaction in rollup. This means that it is possible to access the assets of the sequencer without the burning process of rollup assets.

Impact

If the private key of the `bridge address` or `withdrawer address` is compromised, an attacker could unlock and withdraw assets from the bridge using direct execution. This may lead to a loss of user funds.

Recommendations

Ensure that the private keys of the `bridge address` and `withdrawer address` are securely stored and the users should be aware of and accept this risk.

Remediation

This issue has been acknowledged by Astria.

3.3. Withdrawal event could be reused by bridge admin

Target	astria-bridge-withdrawer		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

In astria-bridge-withdrawer, `bridge address` and `withdrawer address` have permissions to withdraw assets from the bridge.

Both `bridge address` and `withdrawer address` could execute `CollectWithdrawals` and `SubmitWithdrawals` using astria-cli to withdraw assets from the bridge.

But there is no marking or validation to check that the withdrawal has already been spent. This means that the withdrawal event could potentially be reused by the bridge admin to withdraw assets from the bridge multiple times.

Impact

If the private key of the `bridge address` or `withdrawer address` is compromised, an attacker could unlock and withdraw assets from the bridge repeatedly using the same withdrawal event. This may lead to a loss of user funds.

Recommendations

Ensure that the withdrawal event is marked or validated to prevent the reuse of the withdrawal event in the bridge withdrawer.

Remediation

This was remediated in commit 9f618708008e97e63efbe3f1a3a7f31ceb2c39d7 ↗ by changing the using `bridge_account_withdrawal_event_storage_key` function to check if the withdrawal event has already been spent.

3.4. TOCTOU bugs in `ActionHandler`

Target	astria-sequencer/src/transaction/mod.rs		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

The action-handler system in Penumbra works by running the checks of actions in parallel such that side effects of the execution of an action do not affect the checks of another action in the same transaction.

A more concise example of this would be the below call trace:

```
Action 1 -> check_stateless
Action 2 -> check_stateless
...
Action N -> check_stateless


Action 1 -> check_stateful
Action 2 -> check_stateful
...
Action N -> check_stateful


Action 1 -> execute
Action 2 -> execute
...
Action N -> execute
```

This causes issues where developers could expect these checks to act like the transaction handling in Cosmos, which is instead dispatched like this:

```
Action 1 -> check_stateless
Action 1 -> check_stateful
Action 1 -> execute


Action 2 -> check_stateless
Action 2 -> check_stateful
```

```
Action 2 -> execute


Action N -> check_stateless
Action N -> check_stateful
Action N -> execute
```

This mismatch between the intuitive mental model of sequential execution of each action's methods with the actual interleaved order leads to time-of-check time-of-use (TOCTOU) bugs, where Action 1's `execute` modifies the state that was already checked by Action 2's `check_stateful`, and then Action 2's `execute` proceeds as if its `check_stateful`'s checks still held. Mitigating these bugs efficiently can often be done by ensuring that there is a corresponding check in `Transaction::check_stateless` that ensures that multiple actions within the same transaction do not modify the same state.

Impact

Actions such as `BridgeSudoChange` can be executed twice when not expected. Imagine a scenario where the sudo address of the bridge was `0x4141...`.

In one transaction with two messages from `0x4141...`, two bridge sudo change actions are included: the first changing the sudo address to `0x4242...` and the second changing it to `0x4343...`.

It is expected that only the first sudo bridge change action is successful as the second action would no longer be sent by the sudo address. However, with the current action architecture, the second transaction would also be successful.

This applies to all actions; however, it mainly affects actions that rely on state that they themselves change.

Recommendations

Modify the action handling to run the actions in sequence, or be aware of the possibilities and move important checks into `execute`.

Remediation

This was remediated in commit 9f959f4fc492599ffc4a0bfa0d6e29d26b097b4e ↗ by following the penumbra framework fix which is to merge `check_stateless` and `execute` into a combined handler `check_and_execute` such that check and execute always happen sequentially. Introducing a `check_historical` to check historical things.

3.5. Withdrawer address is not working in ICS20 withdrawal

Target	astria-bridge-withdrawer		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The `withdrawer address` has permission to withdraw assets from the bridge on behalf of the `bridge address`.

In astria/crates/astria-sequencer/src/ibc/ics20_withdrawal.rs, the function `check_stateful` checks if the sender of the withdrawal action is not the `bridge address`; the `withdrawer address` should be the sender of the withdrawal action.

```
async fn check_stateful<S: StateReadExt + 'static>(
    &self,
    state: &S,
    from: Address,
) -> Result<()> {
    ics20_withdrawal_check_stateful_bridge_account(self, state, from).await?;
    //...
}

async fn ics20_withdrawal_check_stateful_bridge_account<S: StateReadExt +
    'static>(
    action: &action::Ics20Withdrawal,
    state: &S,
    from: Address,
) -> Result<()> {
    // ...
    let bridge_address = action.bridge_address.unwrap_or(from);

    let Some(withdrawer) = state
        .get_bridge_account_withdrawer_address(&bridge_address)
        .await
        .context("failed to get bridge withdrawer")?
    else {
        bail!("bridge address must have a withdrawer address set");
    };

    ensure!(
```

```
            withdrawer == from,
            "sender does not match bridge withdrawer address; unauthorized"
        );

        Ok(())
    }
```

But the `execute` method uses the `from` address to decrease the balance of the sender. If the sender is `withdrawer address`, the balance of the `withdrawer address` will be decreased instead of the `bridge address`.

```
async fn execute<S: StateWriteExt>(&self, state: &mut S, from: Address) ->
    Result<()> {
    let fee = state
        .get_ics20_withdrawal_base_fee()
        .await
        .context("failed to get ics20 withdrawal base fee")?;
    let checked_packet =
    withdrawal_to_unchecked_ibc_packet(self).assume_checked();

    state
        .decrease_balance(from, self.denom(), self.amount())
        .await
        .context("failed to decrease sender balance")?;

    // ...
}
```

Impact

When the `withdrawer address` tries to withdraw assets from the bridge, it does not work as expected.

Recommendations

Replace the `from` address with the `bridge address` in the `execute` method to decrease the balance of the `bridge address` instead of the `withdrawer address` when the sender is the `withdrawer address`.

Remediation

This was remediated in commit d47a3745a7f3adf8f94d53c86bbcf8378b07be15 ↗ by changing the `establish_withdrawal_target` function return the withdrawal_target address and decrease the balance of the withdrawal_target address instead of the sender address.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. IBC transfer not according to spec

The data packet for an ICS20 transfer is defined as such:

```
interface FungibleTokenPacketData {
  denom: string
  amount: uint256
  sender: string
  receiver: string
  memo: string
}
```

However, the ICS20 transfer code is defined as such, failing if the `packet_data` is too large for a `u128` due to overflow. This is something to be aware of, as a valid ICS20 transfer may fail.

```
#[allow(clippy::too_many_lines)]
async fn execute_ics20_transfer<S: StateWriteExt>(
    state: &mut S,
    data: &[u8],
    source_port: &PortId,
    source_channel: &ChannelId,
    dest_port: &PortId,
    dest_channel: &ChannelId,
    is_refund: bool,
) -> Result<()> {
    let packet_data: FungibleTokenPacketData =
        serde_json::from_slice(data).context("failed to decode
    FungibleTokenPacketData")?;
    let packet_amount: u128 = packet_data
        .amount
        .parse()
        .context("failed to parse packet data amount to u128")?;
    ...
}
```

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the modules and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: lib.rs

`get_for_block_hash`

This is the main function responsible for gathering every log for every action type and calling the respective function for transforming these logs into actions.

It first checks the config to see if every action type should be gathered and sent.

If the action type is configured, it calls `get_logs` with the necessary filter type, specifically in the case that all action types are configured. The filters are `Ics20WithdrawalFilter` and `SequencerWithdrawalFilter`.

However, once the logs are gathered, that is not all; they have to still pass validation. For `ics20` actions, this is done by calling `log_to_ics20_withdrawal_action` on each log and collecting its results. The same is true for `sequencer` actions, and these are transformed, calling the respective function `log_to_sequencer_withdrawal_action`.

Once the logs have been transformed into actions, they are returned.

`get_log`

This function is responsible for actually gathering the logs from blocks. This is done by creating an `ethers::Filter` with the templated event signature, the block address, and the block hash. Then, the provider's `get_logs` function is called with the filter. The returned logs' `removed` parameter is not checked because the chains Astria will connect with have no possibility to re-org.

`log_to_ics20_withdrawal_action`

This function is responsible for validating the logs produced by `astria-smart-contracts`.

```
pub struct Ics20WithdrawalFilter {
    #[ethevent(indexed)]
    pub sender: ::ethers::core::types::Address,
    #[ethevent(indexed)]
    pub amount: ::ethers::core::types::U256,
    pub destination_chain_address: ::std::string::String,
    pub memo: ::std::string::String,
```

```
}
```

These conditions must be true for the log to be valid:

- The log must have a block number.
- The log must have a TX hash.
- The log must be decodable into its indexed topics.
- The log must be serializable to JSON.
- The asset withdrawal division must not fail.

Then the log is transformed into an action with all the logs' indexed topics used as the values for the action.

`log_to_sequencer_withdrawal_action`

This function is responsible for validating the logs produced by `astria-smart-contracts`.

```
pub struct SequencerWithdrawalFilter {
    #[ethevent(indexed)]
    pub sender: ::ethers::core::types::Address,
    #[ethevent(indexed)]
    pub amount: ::ethers::core::types::U256,
    pub destination_chain_address: ::std::string::String,
}
```

These conditions must be true for the log to be valid:

- The log must have a block number.
- The log must have a TX hash.
- The log must be decodable into its indexed topics.
- The log must be serializable to JSON.
- The asset withdrawal division must not fail.
- The log's destination chain address must be parsed as a valid `bech32`. This resulted in Finding 3.1. ↗ because this is a controllable input that would cause the sync and functioning of the bridge withdrawer to stop by erroring out.

Then the log is transformed into an action with all the logs' indexed topics used as the values for the action.

5.2. Module: submitter

`Submitter::run`

This function is responsible for getting processed actions by the watcher thread through the `batches_rx` channel.

It calls `process_batch` on each received batch of actions.

`process_batch`

This is the function responsible for building the transaction that the sequencer will receive with all the actions.

It first gets the next pending nonce, and then it builds an unsigned TX. Following this, it signs the transactions, and then it attempts to submit this TX to the sequencer's CometBFT mempool with an exponential backoff, failing if the TX was not included in the mempool or the block.

It then updates the last rollup height that was submitted and the other state for bookkeeping.

5.3. Module: watcher.rs

`watch_for_blocks`

This is the main watching routine of the bridge withdrawer. This initiates the process of asking a provider (an RPC).

It first subscribes (connects) to a block provider.

It then checks the last synced rollup block height since its start. This is done to sync blocks that were missed between the current `latest` block height on the RPC and the last synced block height.

It then calls `sync_from_next_rollup_block_height`, which is responsible for syncing and watching blocks that were missed.

Once the watcher is synced to the latest block, it loops, gathering new blocks and calling `get_and_send_events_at_block` to process them.

`sync_from_next_rollup_block_height`

This function is responsible for syncing from the last processed block to the latest block from the RPC.

This is done by looping over the block height, asking the provider for said block number, and sequentially calling `get_and_send_events_at_block` on the provided block.

`get_and_send_events_at_block`

This is the main function that processes blocks and monitors them for important cross-chain events that are emitted by `astria-bridge-contracts`.

It first verifies that the block it received is valid, checking that the block structure contains the block hash, block number, and so forth.

It then calls the `action_fetcher.get_for_block_hash`, which retrieves all the actions in the specified block.

If the block contained no important cross-chain events (actions), then it is skipped. Otherwise, it is sent to another thread that is responsible for relaying these messages to the Astria sequencer.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Astria network.

During our assessment on the scoped Astria Bridge modules, we discovered five findings. No critical issues were found. One finding was of high impact, three were of medium impact, and one was of low impact.

6.1. 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.