Aztec: Hatch

Security Review

Cantina Managed review by:
Slowfi, Security Researcher
Arno, Associate Security Researcher

February 24, 2026

Contents

1 Introduction **2**
1.1 About Cantina . . . 2
1.2 Disclaimer . . . 2
1.3 Risk assessment . . . 2
1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
2.1 Scope . . . 3

3 Findings **4**
3.1 High Risk . . . 4
3.1.1 Escape hatch updates can retroactively change historical epoch classification . . . 4
3.2 Medium Risk . . . 4
3.2.1 `BOND_TOKEN`s from taxes and punishments are permanently locked in `EscapeHatch` 4
3.2.2 Updating `EscapeHatch` can invalidate an already selected proposer . . . 4
3.2.3 Inactive escape hatch contracts can still select and punish candidates . . . 5
3.2.4 Slashing round execution can revert when a targeted epoch committee is empty . . . 5
3.2.5 Fee header compression can revert if congestion or prover costs exceed field size . . . 6
3.2.6 Unbounded excess mana can overflow congestion multiplier computation and block proposals . . . 6
3.3 Low Risk . . . 7
3.3.1 NatSpec claims `_slashAmounts` ”must be > 0” but constructor doesn't enforce it . . . 7
3.3.2 Zero bond size allows free participation in escape hatch . . . 7
3.3.3 Escape hatch proposals can skip epoch setup and leave RANDAO checkpoints stale . 7
3.3.4 `EscapeHatch` address can be updated to an incompatible contract . . . 8
3.3.5 Missing explicit bounds/sanity checks for `updateProvingCostPerMana` . . . 8
3.4 Gas Optimization . . . 8
3.4.1 `CandidateJoined` event redundantly emits immutable bond size . . . 8
3.4.2 Candidate bond amount is redundantly stored despite being immutable . . . 9
3.4.3 Proposer index is computed but unused during validator selection . . . 9
3.5 Informational . . . 9
3.5.1 Incorrect NatSpec EIP-712 `Vote` struct field order in `VOTE_TYPEHASH` comment . . . 9
3.5.2 Dead stale-round check in `getRound` . . . 10
3.5.3 `getVotes` can return stale votes for overwritten rounds . . . 10
3.5.4 Incorrect bit-width comment for `CompressedFeeConfig.manaTarget` . . . 10
3.5.5 Dead storage field: unused `feeHeaders` mapping in `FeeLib.FeeStore` . . . 11
3.5.6 Zero or minimal RANDAO lag can allow proposer influence over committee and proposer selection . . . 11
3.5.7 Reward accounting can revert if burn exceeds collected fee . . . 11
3.5.8 Outbox roots can be overwritten without resetting nullifier state . . . 12
3.5.9 Large lag configuration can underflow epoch sample time computation and block early epoch setup . . . 12
3.5.10 Use of magic numbers reduces readability and maintainability . . . 13
3.5.11 Misconfigured committee size can block normal proposal flow . . . 13
3.5.12 `initiateExit()` can self-revert when it selects the caller as proposer . . . 13
3.5.13 Slashing payloads are not epoch-attributable . . . 14
3.5.14 Initial ETH per fee asset can be discarded if ignition checkpoints overwrite fee header with zero . . . 14
3.5.15 Fee asset price modifier unit change can cause proposals to revert if offchain components are not migrated . . . 15
3.5.16 Deployment misconfiguration risk . . . 15

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Aztec Labs was founded in 2017, and has a team of +50 leading zero-knowledge cryptographers, engineers, and business experts. Aztec Labs is developing its namesake products: AZTEC (a privacy-first L2 on Ethereum) and NOIR (the universal ZK language).

From Jan 25th to Jan 27th the Cantina team conducted a review of aztec-packages on commit hash 0092f9f0. The team identified a total of **31** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	6	5	1
Low Risk	5	2	3
Gas Optimizations	3	2	1
Informational	16	6	10
Total	**31**	**16**	**15**

2.1 Scope

The security review had the following components in scope for aztec-packages on commit hash 0092f9f0:

```
l1-contracts/src/core
├── EscapeHatch.sol
├── Rollup.sol
├── RollupCore.sol
├── interfaces
│   ├── IEscapeHatch.sol
│   ├── IRollup.sol
│   └── IValidatorSelection.sol
├── libraries
│   ├── Errors.sol
│   ├── HatchMath.sol
│   ├── compressed-data
│   │   └── fees
│   │       └── FeeConfig.sol
│   └── rollup
│       ├── EpochProofLib.sol
│       ├── FeeLib.sol
│       ├── InvalidateLib.sol
│       ├── ProposedHeaderLib.sol
│       ├── ProposeLib.sol
│       ├── RewardLib.sol
│       ├── ValidatorOperationsExtLib.sol
│       └── ValidatorSelectionLib.sol
├── messagebridge
│   └── Inbox.sol
└── slashing
    └── TallySlashingProposer.sol
```

3 Findings

3.1 High Risk

3.1.1 Escape hatch updates can retroactively change historical epoch classification

Severity: High Risk

Context: EpochProofLib.sol#L293-L298

Description: The function `verifyEpochProof` from library `EpochProofLib` conditionally skips committee attestation verification when the escape hatch reports the hatch is open for the epoch. This decision is made by reading the current escape hatch address and calling `escapeHatch.isHatchOpen(epoch)`.

The rollup does not persist which escape hatch contract was active when a checkpoint was proposed. As a result, governance updates to the escape hatch address can retroactively change how past epochs are interpreted. This impacts multiple security decisions that query the current escape hatch pointer for historical epochs:

- Proof submission can start requiring attestations for a past epoch or stop requiring them, depending on the new escape hatch address and its `isHatchOpen` result.
- Invalidation rules can change for already proposed checkpoints, potentially blocking invalidation that would otherwise be allowed.
- Slashing eligibility can change at tally time, shifting which epochs and actors are considered slashable.

This creates a situation where protocol behavior for a historical epoch depends on a mutable configuration rather than the state that was in effect at proposal time.

Recommendation: Consider to make escape hatch classification epoch stable by snapshotting the escape hatch address or hatch status per epoch. One approach is to activate escape hatch updates at epoch boundaries so that an updated escape hatch becomes effective starting from the next epoch. Another approach is to expose a `getEscapeHatchAt(epoch)` style access pattern backed by a snapshotted history, so proof verification, invalidation, and slashing can consistently use the escape hatch that was active for that epoch.

Aztec Labs: Fixed in PR 20363.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 `BOND_TOKEN`s from taxes and punishments are permanently locked in `EscapeHatch`

Severity: Medium Risk

Context: EscapeHatch.sol#L214

Description: In `EscapeHatch.sol`, `WITHDRAWAL_TAX` is deducted from a candidate's bond refund when they leave the set, and `FAILED_HATCH_PUNISHMENT` is deducted if a proposer fails to fulfill their duties. While these amounts are subtracted from the user's payout, the corresponding `BOND_TOKEN`s remain held by the `EscapeHatch` contract. There is no mechanism to withdraw, burn, or sweep these accumulated funds, causing them to be permanently locked in the contract.

Recommendation: Add a restricted `withdraw` or `sweep` function to allow a governance entity to retrieve accumulated tokens.

Aztec Labs: Acknowledged. This is a design decision.

Cantina Managed: Acknowledged.

3.2.2 Updating `EscapeHatch` can invalidate an already selected proposer

Severity: Medium Risk

Context: RollupCore.sol#L402-L405

Description: The function `updateEscapeHatch` from contract `RollupCore` updates the configured escape hatch address via `updateEscapeHatch` function and emits `EscapeHatchUpdated` without enforcing any timing constraints relative to an in progress hatching cycle at `RollupCore` contract.

If the escape hatch address is changed after a proposer has already been selected in the previous escape hatch instance, the rollup will start interacting exclusively with the new escape hatch contract. The previously selected proposer remains recorded in the old escape hatch contract, but the rollup no longer advances that contract's state. As a result, the proposer can be treated as having failed to propose, even though they were correctly selected and ready to act under the prior escape hatch configuration.

Recommendation: Consider to restrict `updateEscapeHatch` so it can only be executed when no hatching cycle is active. Alternatively, consider to ensure that any proposer already selected under the previous escape hatch remains observable and cannot be penalized until the cycle is cleanly finalized or transitioned.

Aztec Labs: Fixed in PR 20363.

Cantina Managed: Fix verified.

3.2.3 Inactive escape hatch contracts can still select and punish candidates

Severity: Medium Risk

Context: EscapeHatch.sol#L545

Description: The function `selectCandidates` from contract `EscapeHatch` is permissionless and remains callable even after the rollup has switched to a different escape hatch contract.

When governance updates the escape hatch address on the rollup, the previously configured escape hatch contract becomes inactive from the rollup's perspective, but it is not disabled internally. Any account can still call `selectCandidates` on the old contract, transitioning candidates into the proposing state. Since the rollup no longer interacts with that contract, proposals originating from it will not be accepted, and the selected candidates can be punished for failing to propose despite the contract no longer being active.

In addition, there is no mechanism to automatically transition candidates to an exitable state or otherwise protect their bonded funds when the escape hatch is replaced. Candidates who were selected but not yet able to act at the time of the update may remain stuck unless they actively intervene, which relies on timely user behavior rather than protocol guarantees.

Recommendation: Consider to explicitly gate escape hatch operations on whether the contract is currently active for the rollup. For example, consider to prevent `selectCandidates` from progressing state when the escape hatch is no longer the one configured on the rollup, or to allow candidates to exit directly without risk of punishment once the contract becomes inactive. This would reduce reliance on candidate activity and prevent unintended punishment in deactivated escape hatch instances.

Aztec Labs: Fixed in PR 20363.

Cantina Managed: Fix verified.

3.2.4 Slashing round execution can revert when a targeted epoch committee is empty

Severity: Medium Risk

Context: TallySlashingProposer.sol#L919-L928

Description: The function `executeRound` from contract `TallySlashingProposer` builds slashing actions by indexing into `_committees[epochIndex][validatorIndex]` and recording the action.

Slashing votes target past epochs and are encoded as a fixed-size byte array covering `COMMITTEE_SIZE` validator slots per epoch across `ROUND_SIZE_IN_EPOCHS`. The vote encoding does not depend on whether a committee exists for a given targeted epoch, and a quorum can be reached for a slot even if the corresponding epoch has no valid committee.

When calldata is constructed for `executeRound`, an epoch with no committee can only be represented as an empty array for that epoch. If `_committees[epochIndex]` is empty, indexing into `_committees[epochIndex][...]` reverts, causing `executeRound` to fail. This means a single round can become blocked if quorum is reached for any slot in an epoch that does not have a valid committee array.

Recommendation: Consider to defensively skip epochs that do not provide a valid committee array before indexing into `_committees`. This can be done by checking that the committee for the computed epoch index exists and has length `COMMITTEE_SIZE`, and skipping processing for that epoch when it does not. Ex:

```
uint256 epochIndex = i / COMMITTEE_SIZE;

if (escapeHatchEpochs[epochIndex]) continue;
if (_committees[epochIndex].length != COMMITTEE_SIZE) continue;
```

Aztec Labs: Fixed in PR 20361.

Cantina Managed: Fix verified.

3.2.5 Fee header compression can revert if congestion or prover costs exceed field size

Severity: Medium Risk

Context: FeeLib.sol#L149-L168

Description: The function `computeFeeHeader` from library `FeeLib` returns a `FeeHeader` that includes `_congestionCost` and `_proverCost`. These values are later compressed into the on chain fee header representation, where the corresponding fields have fixed bit sizes.

`_congestionCost` and `_proverCost` are computed in fee asset units using a conversion of the form `feeAssetCost` equals `ethCost` times `1e12` divided by `ethPerFeeAsset`. When the fee asset price is low, when L1 fees are high, or when parameters such as the mana target are small, these computed values can grow large enough to exceed the representable range. In that case, fee header compression reverts, which causes checkpoint proposal to revert.

This creates a configuration and market-dependent liveness risk, where checkpoint proposals can fail due to costs exceeding encoding limits rather than being handled as a bounded input.

Recommendation: Consider to enforce explicit upper bounds for `_congestionCost` and `_proverCost` before fee header compression. This can be implemented by clamping to the maximum representable value, or by reverting with a clear error earlier in the flow when costs exceed the allowed range. This would make the encoding constraint explicit and avoid unexpected reverts during compression.

Aztec Labs: Fixed in PR 20362.

Cantina Managed: Fix verified.

3.2.6 Unbounded excess mana can overflow congestion multiplier computation and block proposals

Severity: Medium Risk

Context: FeeLib.sol#L225, FeeLib.sol#L365-L375

Description: The function `congestionMultiplier` from library `FeeLib` computes the congestion multiplier by calling `fakeExponential` with `excessMana` as the numerator.

- `fakeExponential` uses checked arithmetic while iteratively updating the Taylor series terms. As `excessMana` grows, intermediate multiplications in the series can overflow and revert. Since fee computation is part of the checkpoint proposal flow, a revert in the congestion multiplier computation can block checkpoint proposals.
- `excessMana` is derived from prior fee headers and there is no explicit cap applied before using it as the `fakeExponential` numerator, so sustained congestion can push it into ranges where overflow becomes possible.

Recommendation: Consider to bound `excessMana` to a safe maximum before using it in `fakeExponential`, or consider to implement a capped variant of the exponential approximation that saturates to a maximum multiplier instead of reverting. This would avoid proposal liveness depending on `fakeExponential` not overflowing under prolonged congestion.

Aztec Labs: Fixed in PR 20362.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 NatSpec claims `_slashAmounts` "must be > 0" but constructor doesn't enforce it

Severity: Low Risk

Context: TallySlashingProposer.sol#L290

Description: The constructor docs say `_slashAmounts` entries "must be > 0", but the constructor only checks ordering (`_slashAmounts[0] <= _slashAmounts[1] <= _slashAmounts[2]`) and does not `require(_slashAmounts[i] > 0)`. As written, zero slash amounts are allowed.If any amount is `0`, slashing can reach quorum but slash nothing. On the other hand, if any amount exceeds `uint96`, `executeRound` will revert when building the payload, permanently disabling slashing.

```
SLASH_AMOUNT_SMALL = _slashAmounts[0];
SLASH_AMOUNT_MEDIUM = _slashAmounts[1];
SLASH_AMOUNT_LARGE = _slashAmounts[2];
// ...
require(_slashAmounts[0] <= _slashAmounts[1],
↪  Errors.TallySlashingProposer__InvalidSlashAmounts(_slashAmounts));
require(_slashAmounts[1] <= _slashAmounts[2],
↪  Errors.TallySlashingProposer__InvalidSlashAmounts(_slashAmounts));
```

Recommendation: Add explicit validation (e.g., `require(_slashAmounts[i] > 0)` for all 3).

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.3.2 Zero bond size allows free participation in escape hatch

Severity: Low Risk

Context: EscapeHatch.sol#L124

Description: The function `constructor` from contract `EscapeHatch` assigns `BOND_SIZE = _bondSize` without explicitly validating that `_bondSize > 0`. If the contract is deployed with `_bondSize == 0`, any address can join the escape hatch set at zero cost while remaining eligible for selection. This removes the intended economic gating described in the documentation and weakens the assumptions around the escape hatch mechanism.

Recommendation: Consider to add an explicit constructor check that `_bondSize` is greater than zero, so a misconfigured deployment cannot silently disable the bond requirement.

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.3.3 Escape hatch proposals can skip epoch setup and leave RANDAO checkpoints stale

Severity: Low Risk

Context: ProposeLib.sol#L202-L205

Description: The function `propose` from library `ProposeLib` sets up the epoch by calling `ValidatorSelectionLib.setupEpoch(v.currentEpoch)` after deriving the current epoch from `block.timestamp`.

In the updated logic, escape hatch proposals do not call `setupEpoch`. This means that for escape hatch epochs the protocol may not refresh epoch specific randomness checkpoints and related epoch initialization state. As a result, subsequent epochs may derive selection randomness from an older checkpoint than intended, making the randomness effectively known earlier and reducing the intended unpredictability for committee and proposer selection.

This is particularly relevant because epoch setup is currently triggered by the first checkpoint of the epoch. If the first checkpoint is proposed through the escape hatch path and epoch initialization is skipped, the refresh may never occur for that epoch.

Recommendation: Consider to ensure the randomness checkpoint is refreshed even when the first checkpoint of an epoch is proposed through the escape hatch path, while still avoiding committee sampling. One approach is to checkpoint the RANDAO for the current epoch during escape hatch proposals and call full epoch setup during non-escape hatch proposals.

Aztec: Acknowledged. It is a minor potential issue, but it is possible to checkpoint the randao whenever desired.

Cantina Managed: Acknowledged.

3.3.4 `EscapeHatch` address can be updated to an incompatible contract

Severity: Low Risk

Context: RollupCore.sol#L402-L405

Description: The function `updateEscapeHatch` from contract `RollupCore` updates the escape hatch address by calling `ValidatorOperationsExtLib.updateEscapeHatch(_escapeHatch)` and emits `EscapeHatchUpdated`.

The update does not validate that the new escape hatch contract is correctly configured to work with this rollup instance. If governance sets an address that is not wired to this rollup, escape hatch proposals can fail or cause rollup interactions with the escape hatch to revert, including calls that assume a compatible interface and correct rollup linkage.

This is primarily a governance configuration risk, but the failure mode can impact liveness for the escape hatch path and create operational risk during upgrades.

Recommendation: Consider to validate the new escape hatch during `updateEscapeHatch` before applying it. This can be done by requiring that the new contract reports it is configured for this rollup, or by performing a minimal compatibility check that exercises the expected interface and confirms the rollup linkage, then reverting if the check fails.

Aztec: Acknowledged. It requires a bad governance proposal, and can be undone.

Cantina Managed: Acknowledged.

3.3.5 Missing explicit bounds/sanity checks for `updateProvingCostPerMana`

Severity: Low Risk

Context: FeeLib.sol#L111-L129

Description: `FeeLib.updateProvingCostPerMana` updates `FeeConfig.provingCostPerMana` with no explicit domain validation (unlike `manaTarget`, which is validated via `computeManaLimit`). The only effective constraint is **implicit**: when recompressing the config, `provingCostPerMana` is downcast to **`uint64`** (`toUint64()`), so values above `type(uint64).max` revert, but no "sane range" bound is enforced.

Recommendation: Add an explicit upper bound (and optionally a lower bound) for `_provingCostPerMana` consistent with intended economics, similar in spirit to `computeManaLimit` for `manaTarget`.

Aztec Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Gas Optimization

3.4.1 `CandidateJoined` event redundantly emits immutable bond size

Severity: Gas Optimization

Context: EscapeHatch.sol#L156

Description: The function `join` from contract `EscapeHatch` emits the `CandidateJoined` event with `BOND_SIZE` as an argument.

Since `BOND_SIZE` is an immutable value set at construction time, emitting it on every `CandidateJoined` event does not convey new information. Indexers and off-chain consumers can already derive the bond size directly from the contract configuration, making this event field redundant.

Recommendation: Consider to remove `BOND_SIZE` from the `CandidateJoined` event to reduce redundancy and simplify event consumption.

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.4.2 Candidate bond amount is redundantly stored despite being immutable

Severity: Gas Optimization

Context: EscapeHatch.sol#L152

Description: The function `join` from contract `EscapeHatch` assigns `data.amount = BOND_SIZE` when a candidate joins the set. Since `BOND_SIZE` is an immutable value shared by all candidates, storing the same bond amount per candidate is redundant and increases storage usage without adding expressiveness. The value never diverges per user unless modified by later punishment logic, which is not currently reflected in the stored structure.

This design also limits flexibility in how penalties are represented, as the full bond amount is always stored even though only partial deductions may apply during the candidate lifecycle.

Recommendation: Consider to avoid storing the full bond amount per candidate when it is invariant. An alternative approach is to store only the penalty applied to a candidate when they fail to propose valid checkpoints, and derive the withdrawable amount at exit by subtracting the accumulated penalty and the withdrawal tax from `BOND_SIZE`. This would preserve correctness while reducing redundant storage and making penalty application more explicit.

Aztec: Acknowledged. Logic simple to follow this way.

Cantina Managed: Acknowledged.

3.4.3 Proposer index is computed but unused during validator selection

Severity: Gas Optimization

Context: ValidatorSelectionLib.sol#L330

Description: The function that derives proposer information from library `ValidatorSelectionLib` computes a `proposerIndex` using `computeProposerIndex` but the computed value is not subsequently used.

This introduces dead logic in the selection flow and makes it unclear whether proposer selection is intended to rely on this value or whether the computation is a leftover from an earlier design. Leaving unused selection logic in place increases maintenance burden and can cause confusion when reasoning about proposer selection correctness.

Recommendation: Consider to either remove the unused `proposerIndex` computation or explicitly use it as part of proposer selection if it is intended to affect protocol behavior. Clarifying this intent in code will improve readability and reduce the risk of incorrect assumptions in future changes.

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Incorrect NatSpec EIP-712 `Vote` struct field order in `VOTE_TYPEHASH` comment

Severity: Informational

Context: TallySlashingProposer.sol#L154-L156

Description: In `TallySlashingProposer.sol`, the NatSpec for `VOTE_TYPEHASH` states the EIP-712 struct as `Vote(uint256 slot,bytes votes)`, but the actual type hash is computed from

`keccak256("Vote(bytes votes,uint256 slot)")`, i.e. the field order is `votes` then `slot`.This is a documentation-only mismatch.

Recommendation: Update the NatSpec to reflect the correct EIP-712 struct definition: `Vote(bytes votes,uint256 slot)`.

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.5.2 Dead stale-round check in `getRound`

Severity: Informational

Context: TallySlashingProposer.sol#L571

Description: In `getRound`, the guard `if (roundData.roundNumber != _round)` is **unreachable**. `_getRoundData` always returns a `RoundData` with `roundNumber: _round`, even when the circular buffer contains data for a different (overwritten) round (it returns a zeroed struct but still sets `roundNumber = _round`).

Recommendation: Remove the `roundData.roundNumber != _round` branch and rely on `_getRoundData`'s default `(executed=false, voteCount=0)` behavior.

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.5.3 `getVotes` can return stale votes for overwritten rounds

Severity: Informational

Context: TallySlashingProposer.sol#L594

Description: `roundVotes` is stored in a circular buffer (`ROUNDABOUT_SIZE`), indexed by `round % ROUNDABOUT_SIZE`. When the buffer wraps, older rounds' vote slots are overwritten/reused. Staleness detection exists in `_getRoundData` via `roundDatas[...]` (compressed `roundNumber`), but `getVotes` does not call it and instead reads vote slots directly, so callers can receive vote data that actually belongs to a different (newer) round.

Recommendation: In `getVotes`, validate round freshness before reading votes (e.g., call `_getRoundData(_round, getCurrentRound())` and/or check `roundDatas[idx].roundNumber.decompress() == _round`). If stale, revert or return empty bytes.

Aztec Labs: Fixed in PR 20756.

Cantina Managed: Fix verified.

3.5.4 Incorrect bit-width comment for `CompressedFeeConfig.manaTarget`

Severity: Informational

Context: FeeConfig.sol#L25

Description: In `FeeConfig.sol`, the comment says `manaTarget` is **64 bits**, but the compression uses `toUint32()` and extraction masks with `MASK_32_BITS`, meaning `manaTarget` is 32 bits in `CompressedFeeConfig`.

Recommendation: Update the comment to reflect the actual layout, e.g. "32 bit `manaTarget`, 128 bit `congestionUpdateFraction`, 64 bit `provingCostPerMana`".

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.5.5 Dead storage field: unused `feeHeaders` mapping in `FeeLib.FeeStore`

Severity: Informational

Context: FeeLib.sol#L65

Description: `FeeLib.FeeStore` declares `feeHeaders` but it is never read from or written to anywhere in the codebase. Fee header lookups instead use `STFLib.getFeeHeader(...)`, making this mapping dead code/storage.

Recommendation: Remove `feeHeaders` from `FeeStore`.

Aztec Labs: Fixed in PR 20423.

Cantina Managed: Fix verified.

3.5.6 Zero or minimal RANDAO lag can allow proposer influence over committee and proposer selection

Severity: Informational

Context: ValidatorSelectionLib.sol#L138-L151

Description: The function `initialize` from library `ValidatorSelectionLib` stores `_lagInEpochsForRandao` without enforcing a minimum value beyond the requirement that `_lagInEpochsForValidatorSet` is greater than or equal to `_lagInEpochsForRandao`.

If `_lagInEpochsForRandao` is configured as zero, the randomness used for selection can depend on `block.prevrandao` from the current epoch context. This gives the current epoch proposer more opportunity to influence the randomness input and bias committee selection and escape hatch proposer selection.

If `_lagInEpochsForRandao` is configured equal to `_lagInEpochsForValidatorSet`, the design still permits minimal separation between the selected validator set and the randomness used to select from it, which can reduce the intended unpredictability and increase the value of proposer influence.

While this is primarily a deployment configuration risk, the impact is security-relevant because it affects the integrity of validator committee and proposer selection.

Recommendation: Consider to enforce a minimum value for `_lagInEpochsForRandao`, such as requiring it to be at least one epoch, and consider to require that `_lagInEpochsForValidatorSet` is strictly greater than `_lagInEpochsForRandao` if the protocol relies on separation between the validator set snapshot and the randomness snapshot. This would prevent misconfiguration that makes selection more biasable.

Aztec: Acknowledged. Used potentially low values to speed up testing, but for a real deployment will be using larger values.

Cantina Managed: Acknowledged.

3.5.7 Reward accounting can revert if burn exceeds collected fee

Severity: Informational

Context: RewardLib.sol#L235-L238

Description: The function that accounts rewards from contract `RewardLib` computes the prover and sequencer fees by subtracting `burn` from `fee`. The logic assumes that `fee` is always greater than or equal to `burn`. If this assumption is violated, for example due to malformed proof inputs or an upstream circuit bug, the expression `fee - burn` underflows and causes proof submission to revert. This introduces a hard failure mode in reward accounting rather than a controlled rejection with a clear error.

While this situation may not be expected under correct circuit behavior, the assumption is implicit and not enforced at the contract level.

Recommendation: Consider to add an explicit validation that `fee` is greater than or equal to `burn` before performing the subtraction, and revert with a clear error if the invariant is violated. This would make the assumption explicit and improve robustness against unexpected inputs.

Aztec: Acknowledged. Should be impossible to hit, unless there issues in the circuits.

Cantina Managed: Acknowledged.

3.5.8 Outbox roots can be overwritten without resetting nullifier state

Severity: Informational

Context: Outbox.sol#L47-L56

Description: The function `insert` from contract `Outbox` writes `roots[_checkpointNumber].root = _root` at `l1-contracts/src/core/messagebridge/Outbox.sol:47` and emits `RootAdded`. The function only checks that the caller is the rollup and that `_checkpointNumber` is greater than the rollup proven checkpoint number.

There is no guard preventing reinsertion for the same checkpoint number. If the rollup ever calls `insert` again for a checkpoint number that already has a stored root, the root can be overwritten while any message consumption state that depends on the previous root remains unchanged. In particular, if messages were already consumed under the old root, the associated nullifier bitmap is not reset or migrated, which can block messages under the new root or desynchronize consumption state from the active root.

This relies on a protocol assumption that outbox roots are written once and never updated, and that leaf identifiers remain stable. The contract does not enforce this assumption.

Recommendation: Consider to enforce one-time insertion per checkpoint number by reverting if a root is already set for `_checkpointNumber`. If root updates are intended to be supported, consider to define and implement explicit state transition logic that keeps nullifier tracking consistent across root changes.

Aztec: Acknowledged. Allowing multiple writes without rewriting the nullifiers is intentional. The nullifiers can only be written to if the root was proven, and at that point the rollup should not overwrite it again. But if a prune happens (lack of proof) then we might need to rewrite the root. This component also altered in hatch 2 sections (indiretly at least) because of the outhash changes.

Cantina Managed: Acknowledged.

3.5.9 Large lag configuration can underflow epoch sample time computation and block early epoch setup

Severity: Informational

Context: ValidatorSelectionLib.sol#L674-L692

Description: The functions `stableEpochToRandaoSampleTime` and `stableEpochToValidatorSetSampleTime` from library `ValidatorSelectionLib` compute a sample timestamp by subtracting `lagInEpochs` multiplied by `epochDuration` from the epoch start timestamp, using `uint32` arithmetic.

If `lagInEpochsForRandao` or `lagInEpochsForValidatorSet` is configured too large relative to the genesis time and the early epoch start timestamps, the subtraction underflows and reverts. This can prevent `setupEpoch` and other functions that depend on these sample time computations, such as `getSampleSeed`, from working during early epochs after deployment.

This is primarily a configuration risk, but the failure mode is a hard revert that can block epoch setup.

Recommendation: Consider to add initialization time validation that the genesis time and configured lags are compatible with the epoch duration. One approach is to ensure that the genesis time offset is at least `lagInEpochs` multiplied by `epochDuration`, or otherwise enforce bounds on the configured lags to prevent underflow in early epochs.

Aztec: Acknowledged. The lag would need to be VERY large for this to happen as the underflow must be with current time, so won't be fixed as that kinda delay would anyway mean that the rollup also has delays of ~50 years from entry to usage.

Cantina Managed: Acknowledged.

3.5.10 Use of magic numbers reduces readability and maintainability

Severity: Informational

Context: EscapeHatch.sol#L184, TallySlashingProposer.sol#L957

Description: The contract `EscapeHatch` computes the next target hatch using the literal value `1` in arithmetic with `LAG_IN_HATCHES` at `l1-contracts/src/core/EscapeHatch.sol:184`, and a similar literal is used again shortly after. The meaning of this increment is implicit and not documented in code.

Similarly, the contract `TallySlashingProposer` allocates a fixed-size `bytes` array using the literal expression `4 * 32` at `l1-contracts/src/core/slashing/TallySlashingProposer.sol:957`. The significance of this length is not encoded in a named constant, making it less clear what structure or expectation the value represents.

In both cases, the use of raw numeric literals makes the code harder to reason about and more error-prone during future modifications.

Recommendation: Consider to replace these literal values with named constants that reflect their semantic meaning. This would improve readability, reduce the risk of accidental misuse, and make future changes easier to apply safely.

Aztec: Acknowledged. The 1 generally used for the next, and the votes size is easily follow for the 4 slots.

Cantina Managed: Acknowledged.

3.5.11 Misconfigured committee size can block normal proposal flow

Severity: Informational

Context: ValidatorSelectionLib.sol#L648-L651

Description: The function that validates committee sampling from library `ValidatorSelectionLib` requires `validatorSetSize` to be greater than or equal to `targetCommitteeSize`, and reverts otherwise.

If governance configures `targetCommitteeSize` on `RollupCore` to a value greater than the size of the active validator set, normal proposal paths that depend on committee formation will revert. This can block committee queries and standard checkpoint proposals, effectively forcing progress to rely on the escape hatch path.

This behavior relies on correct governance configuration and does not provide a graceful degradation or early validation when the configuration becomes incompatible with the validator set size.

Recommendation: Consider to validate committee size configuration changes at the time they are applied, ensuring that `targetCommitteeSize` does not exceed the current validator set size. Alternatively, consider to define explicit behavior for this case, such as clamping the effective committee size or preventing configuration updates that would block the normal proposal flow.

Aztec: Acknowledged. The configuration would need to be specified at deployment of the rollup and then added as the new rollup to take effect. But if that is the case, yes it could stall forever. However, as those kinda of stalls are also possible in other cases where rollup is updated to broken code etc, it don't seems particularly likely. It relies on no-one validating and if done maliciously worse things could happen.

Cantina Managed: Acknowledged.

3.5.12 `initiateExit()` can self-revert when it selects the caller as proposer

Severity: Informational

Context: EscapeHatch.sol#L169

Description: `initiateExit()` calls `selectCandidates()` first (`EscapeHatch.sol:169`). If that internal call selects `msg.sender` as the designated proposer, it updates the caller's state to `PROPOSING` and removes them from `$activeCandidates`. Control then returns to `initiateExit()`, which immediately requires the caller is still in `$activeCandidates` and has `Status.ACTIVE`, so the transaction reverts (typically with a misleading `NotInCandidateSet/InvalidStatus`).

This makes the behavior/commentary ambiguous: the inline comment suggests `selectCandidates()` just "simplifies" subsequent checks, but it can also mutate state such that those checks intentionally fail. Separately, the `selectCandidates()` comment about handling `Status.EXITING` is subtle: it only applies when a candidate initiated exit in an earlier transaction before the selection window for that hatch, and is later selected from the snapshot (not within the same `initiateExit()` call).

Recommendation:

- Update NatSpec/comments to explicitly state that a candidate cannot initiate exit if they become the designated proposer; they must follow the `PROPOSING -> validateProofSubmission -> EXITING -> leaveCandidateSet` flow.
- Consider adding an explicit post-`selectCandidates()` check that reverts with a dedicated error (e.g., "selected proposer cannot exit") instead of relying on the later membership/status `require`s, to avoid confusing revert reasons.

Aztec Labs: Fixed in PR 20363.

Cantina Managed: Fix verified.

3.5.13 Slashing payloads are not epoch-attributable

Severity: Informational

Context: TallySlashingProposer.sol#L920-L922

Description: `TallySlashingProposer` tallies votes over a flattened set of committee positions across all epochs in the round (`COMMITTEE_SIZE * ROUND_SIZE_IN_EPOCHS`) and converts any position that reaches quorum into a `SlashAction` by mapping the position index `i` to an address via `_committees[i / COMMITTEE_SIZE][i % COMMITTEE_SIZE]`.

Because actions are created per position and there is no de-duplication by validator address, the same validator address can appear multiple times in the resulting `actions[]` if it appears in multiple epoch committees within the same slashing round. Each action becomes a separate `IStakingCore.slash(validator, amount)` call via `SlashPayloadCloneable`, which encodes only `(validator, amount)` and carries no epoch/offense identifier.

As a result, the onchain execution path cannot distinguish whether a validator is being slashed for epoch 0 vs epoch 1 (or any specific offense); it only reflects that the validator was slashed one or more times. Any intended policy like "slash multiple times only if they offended multiple times (in different epochs)" is therefore not enforceable onchain and relies on proposers voting correctly per position rather than "blanket voting" across all appearances.

Recommendation: If epoch attribution matters (e.g., to ensure "only slash for the specific epoch(s) of misbehavior").

Aztec Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.5.14 Initial ETH per fee asset can be discarded if ignition checkpoints overwrite fee header with zero

Severity: Informational

Context: STFLib.sol#L119-L133

Description: The function `writeGenesisFeeHeader` from contract `STFLib` writes the configured `_initialEthPerFeeAsset` into checkpoint 0. However, if checkpoints are proposed while transactions are disabled during ignition with `manaTarget == 0`, the propose path can skip fee header computation and leave the proposed checkpoint fee header with `ethPerFeeAsset == 0`. This can overwrite the nonzero genesis value.

When transactions are later enabled and the first checkpoint with transactions is proposed, the function `computeFeeHeader` from contract `FeeLib` reads the parent fee header and applies `Math.max(parentEthPerFeeAsset, MIN_ETH_PER_FEE_ASSET)` to handle ignition checkpoints where `ethPerFeeAsset` may be zero. In this situation, the fee update seeds from `MIN_ETH_PER_FEE_ASSET` rather than the configured `_initialEthPerFeeAsset`. Since the per checkpoint adjustment is capped to

plus minus 1 percent, the fee can remain far below the intended level for many checkpoints after ignition, resulting in sustained underpricing until it slowly drifts back toward a realistic value.

This is avoided only if no checkpoints are proposed during ignition and `manaTarget > 0` is enabled before the first proposed checkpoint.

Recommendation: Consider to ensure that ignition checkpoints cannot overwrite `ethPerFeeAsset` with zero once the genesis value is set. Consider to copy the parent fee header forward during ignition proposals, or explicitly preserve `ethPerFeeAsset` even when fee header computation is skipped, so that the first checkpoint with transactions seeds pricing from the configured initial value rather than `MIN_ETH_PER_FEE_ASSET`.

Aztec: Acknowledged, this is acceptable and expected. The current setup will likely not be going from 0 to something without also needing changes to the verification keys or the like, which require a full new deployment. So it is not expected that the same instance will both have 0 and something, but the code is allowing it to support easily having 2 separate.

Cantina Managed: Acknowledged.

3.5.15 Fee asset price modifier unit change can cause proposals to revert if offchain components are not migrated

Severity: Informational

Context: FeeLib.sol#L184-L205

Description: The function `computeFeeHeader` from contract `FeeLib` now enforces that `_feeAssetPriceModifierBps` is within `MAX_FEE_ASSET_PRICE_MODIFIER_BPS`, and the parameter name indicates the value is expected in basis points. However, the corresponding field naming and type in surrounding interfaces and offchain payloads can remain unchanged from the previous scale.

If an offchain component continues sending the modifier using the previous scale, the absolute value can exceed `MAX_FEE_ASSET_PRICE_MODIFIER_BPS` and the call will revert with `FeeLib__InvalidFeeAssetPriceModifier`. Since fee header computation is part of the propose flow, this can prevent proposals from being accepted and halt checkpoint progression until all offchain producers are migrated to the new basis points convention.

Recommendation: Consider to make the basis points unit expectation unambiguous at the interface boundary by renaming the field and related interface parameter to include `Bps` consistently, and updating any encoded struct or message schema names accordingly. Consider to add a temporary compatibility path that accepts the previous scale during a migration window, or emit a clear event or revert reason that includes the received value and the expected maximum to simplify operational debugging.

Aztec: Acknowledged, keeping it as it is.

Cantina Managed: Acknowledged.

3.5.16 Deployment misconfiguration risk

Severity: Informational

Context: FeeLib.sol#L115-L128

Description: The function `initialize` from contract `FeeLib` validates that `_initialEthPerFeeAsset` is within `MIN_ETH_PER_FEE_ASSET` and `MAX_ETH_PER_FEE_ASSET`, but it assumes the value is provided in the correct fixed point scale. If the deployment configuration sets `AZTEC_INITIAL_ETH_PER_FEE_ASSET` with an incorrect order of magnitude, the value can still pass bounds checks while representing a materially incorrect starting price.

This can lead to the fee market initializing at a significantly mispriced level, affecting early checkpoints until the pricing mechanism gradually adjusts.

Recommendation: Consider to explicitly document the expected fixed point scale and magnitude for `AZTEC_INITIAL_ETH_PER_FEE_ASSET` in deployment configuration and tooling. Consider to add a deployment time sanity check or helper that validates the configured value against an expected human readable range to reduce the risk of misconfiguration.

Aztec Labs: Acknowledged.

Cantina Managed: Acknowledged.