Aztec PR 20865

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher

March 2, 2026

Contents

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

A security review is 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 the review 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 a 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 Feb 27th to Feb 28th the security researchers conducted a review of aztec-packages on commit hash ed3f61d6. A total of **5** issues were identified:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	4	0	4
Total	**5**	**1**	**4**

2.1 Scope

The security review had the following components in scope for aztec-packages on commit hash ed3f61d6:

```
l1-contracts
├── script
│   └── deploy
│       └── DeployAlpha.s.sol
├── scripts
│   └── run_alpha_upgrade.sh
└── src
    └── alpha
        ├── AlphaPayload.sol
        └── AlphaVerifier.sol
```

3 Findings

3.1 High Risk

3.1.1 Malformed committees bypass slashing execution

Severity: High Risk

Context: TallySlashingProposer.sol#L465-L490, TallySlashingProposer.sol#L907-L917

Description: `TallySlashingProposer` currently mixes two behaviors that are individually understandable but collectively create a fail-open slashing bypass. In commit b7605403131 dated 2026-02-09, the change titled ”fix: skip empty epochs in tally slasher” introduced `test_executeRoundWithEmptyCommittee` to prevent execution reverts when early epochs naturally have no sampled committee yet. That intent is valid for liveness because naturally unavailable committee data in bootstrap windows should not brick `executeRound`.

The problem is that the same logic also accepts attacker-supplied malformed committees in normal slashable windows. `executeRound` is permissionless and accepts caller-provided `_committees` as long as the outer array length matches `ROUND_SIZE_IN_EPOCHS`. During tally, `_determineSlashActions` skips an epoch whenever `_committees[epochIndex].length != COMMITTEE_SIZE`. This skip happens before any slashing action is created for that epoch, so `committeesWithSlashes[epochIndex]` remains false even if vote data for validators in that epoch is nonzero and quorum would otherwise be met.

After tally, `executeRound` verifies committee commitments only for indices where `committeesWithSlashes` is true. If the caller supplies malformed inner arrays for targeted epochs, no actions are produced, no commitment checks run and the function still sets `executed = true`. Because `RoundAlreadyExecuted` then blocks re-execution, a single permissionless call can permanently finalize that round with zero slashes. The existing test demonstrates acceptance of empty committees for early unavailable epochs, but it does not establish that accepting arbitrary malformed committees in slashable epochs is safe. It therefore confirms partial intent, not full safety.

A realistic failure path is straightforward. A round becomes executable after the delay window. Honest proposers have already cast slash votes that should meet quorum for one or more validators. Any external caller submits `executeRound` with correctly sized outer `_committees` and empty or wrong-sized inner arrays for those epochs. Tally silently drops those epochs, action count becomes zero, the round is marked executed, and expected penalties can no longer be applied for that round.

Impact is loss of slashing effectiveness for affected rounds and a potential permanent penalty bypass for slashable behavior.

Proof of concept: Realistic Forge proof of concept test added in test file test/slashing/TallySlashingProposer.t.sol:397:

- `Test_executeRoundBypassesSlashingWithMalformedCommitteeInput`.

What it proves end-to-end:

1. With valid committees, quorum votes produce a real slash action.
2. A permissionless attacker executes the same round with malformed inner committees (`[]` for slashable epoch).
3. Round is marked executed anyway.
4. Target validator balance remains unchanged (no slash happened).
5. Re-execution with valid committees reverts with `TallySlashingProposer__RoundAlreadyExecuted`.
6. So slashing is permanently bypassed for that round.

Recommendation: Adopt an explicit fail-closed policy by removing the skip path for malformed committee length in `_determineSlashActions`. This directly prevents a caller from converting slashable votes into zero actions and then finalizing the round as executed. Under this policy, quorum on invalid or unavailable committee data is treated as invalid execution input and the round reverts instead of silently succeeding.

The concrete change is to delete this branch:

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

After removing it, indexing `_committees[epochIndex][i % COMMITTEE_SIZE]` naturally reverts when callers submit malformed inner arrays, which blocks the fail-open finalize path.

This recommendation intentionally accepts the liveness tradeoff that originally motivated the skip-path fix. If quorum is cast for committee slots that are unavailable or nonsensical, execution can revert until a valid execution context is provided or the round expires. In other words, the system chooses safety over permissive progress for this edge case.

Aztec Labs: Fixed in PR 20987.

R0bert: Fix verified.

3.2 Informational

3.2.1 Alpha upgrade configuration changes and purpose

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: This note records the configuration deltas introduced by the alpha upgrade, comparing the current canonical mainnet state to the new rollup and governance payload configuration. The goal is change-control clarity: each listed item includes what changes and why it exists in the upgrade design. The material configuration changes in this upgrade are:

Canonical rollup pointer:

- Before: canonical rollup = `0x603bb2c05D474794ea97805e8De69bCcFb3bCA12`.
- After: canonical rollup moves to the newly deployed alpha rollup.
- Purpose: cut over execution and governance to the new rollup implementation and storage domain.

Committee target size:

- Before: 24.
- After: 48.
- Purpose: increase committee width and attestation participation target for the upgraded regime.

Local ejection threshold:

- Before: 196,000e18.
- After: 190,000e18.
- Purpose: relax rollup-local post-slash ejection strictness relative to the prior local threshold.

Mana target:

- Before: 0.
- After: 75,000,000.
- Purpose: transition from ignition mode into active tx-fee mode by enabling nonzero mana throughput.

Proving cost per mana:

- Before: 0.
- After: 25,000,000 wei.
- Purpose: introduce a nonzero proving-cost component in fee computation.

Checkpoint reward:

- Before: `400e18`.
- After: `500e18`.

- Purpose: raise checkpoint-level reward emissions for sequencer/prover incentive budget in the new phase.

Governance execution delay:

- Before: 604,800 seconds (7 days).
- After: 2,592,000 seconds (30 days).
- Purpose: lengthen governance timelock between queue and executable states.

Verifier implementation pinning:

- Before: epoch proof verifier runtime hash =

  [] 0x5815ebdc2b97702193ed35ce4caec0a17312c16cb6b3d2509688d6ac424323c1
- After: expected verifier runtime hash =

  [] 0x9a0aed515ad9e25d127fc25746b81a92701c2113f894f1122d87d32d98569e28
- Purpose: swap proving verifier implementation to the alpha verifier binary.

Genesis VK tree root (new rollup state):

- Before: `0x229eadb7c540c82204b5373633d3c25557f8264ad8fca760660fe853e5275e39`.
- After: `0x2d0b15497929f5150c4c383993555456e60d27121f4ac2cb9ef880319f5f9a6f`.
- Purpose: pin upgraded proving key commitment for the new rollup genesis.

Protocol contracts hash (new rollup state):

- Before: `0x12e9aa367b065eff3e48912b8cae62209970117d34a8c9ef1e9e4116e41bc8d6`.
- After: `0x2672340d9a0107a7b81e6d10d25b854debe613f3272e8738e8df0ca2ff297141`.
- Purpose: pin upgraded protocol-contract commitment for new rollup genesis.

Escape hatch activation on new rollup:

- Before: new rollup pre-execution escape hatch is unset.
- After: payload sets new escape hatch contract.
- Purpose: enable last-resort liveness/fallback mechanism only after governance execution.

Rewards-claimable flag on new rollup:

- Before: new rollup deploys with rewards claiming disabled.
- After: payload enables rewards claiming.
- Purpose: align reward claimability with post-upgrade operational state.

Some other settings are still included in the new rollup configuration hash because this is a new rollup instance, but their numbers are intentionally the same as before (the current deployed version).

Recommendation: Merely informative. Use this change log as a mandatory sign-off artifact for governance and operations prior to payload execution. Pair it with a rollout checklist that validates runtime flags and node behavior are consistent with the new on-chain fee mode and require explicit approval for each economic parameter delta before cutover.

Aztec Labs: Acknowledged.

R0bert: Acknowledged.

3.2.2 Transaction defaults block post-activation flow

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The runtime configuration currently combines a tx-enabled on-chain fee regime with off-chain defaults that still enforce tx-disabled behavior. After the upgrade, `manaTarget` is nonzero and the fee model expects a normal transaction-bearing flow, but the node defaults still disable transactions at

the networking and sequencing layers. This creates an inconsistent operating mode where L1 accepts tx economics while the operator stack can continue rejecting or never propagating user transaction traffic.

This conflicts with the public Participate docs, which describe a normal post-activation transaction lifecycle where users submit transactions to the network, sequencers include those transactions in blocks and users pay fees for those transactions in `AZTEC` (https://docs.aztec.network/participate/basics/transactions and https://docs.aztec.network/participate/basics/fees). The governance docs also describe upgrade and proposal flows that assume continued network operation and execution capacity across versions (https://docs.aztec.network/participate/governance/proposal-lifecycle and https://docs.aztec.network/participate/governance/upgrades).

The mainnet default environment sets transaction suppression toggles that are appropriate for ignition, but risky after fee activation. The configuration explicitly disables transactions, sets max transactions per block to zero, and sets pending transaction pool capacity to zero.

```
# spartan/environments/network-defaults.yml
TRANSACTIONS_DISABLED: true
SEQ_MAX_TX_PER_BLOCK: 0
P2P_MAX_PENDING_TX_COUNT: 0
```

The P2P service conditionally removes tx-related request-response handlers when transaction mode is disabled, which suppresses tx transport behavior even if peers submit traffic.

```
// yarn-project/p2p/src/services/libp2p/libp2p_service.ts
if (!this.config.disableTransactions) {
  requestResponseHandlers[ReqRespSubProtocol.BLOCK_TXS] = blockTxsHandler.bind(this);
}

if (!this.config.disableTransactions) {
  requestResponseHandlers[ReqRespSubProtocol.TX] = txHandler.bind(this);
}
```

Proposal validation also rejects proposals that carry transactions when transaction mode is disabled. This means even if a tx-bearing proposal appears, peers configured in disabled mode can classify it as invalid behavior.

```
// yarn-project/p2p/src/msg_validators/proposal_validator/proposal_validator.ts
const embeddedTxCount = proposal.txs?.length ?? 0;
if (!this.txsPermitted && (proposal.txHashes.length > 0 || embeddedTxCount > 0)) {
  return { result: 'reject', severity: PeerErrorSeverity.MidToleranceError };
}
```

In practical terms, the network can remain tx-closed due to operator defaults even though the rollup has moved into a tx-priced state.

Recommendation: Treat transaction-mode alignment as a release gate for the upgrade process. The deployment checklist should require explicit confirmation that transaction suppression toggles are switched to tx-enabled values before or exactly at cutover, including `TRANSACTIONS_DISABLED=false`, nonzero tx-per-block capacity and nonzero pending tx pool limits.

Aztec Labs: Acknowledged. This is a work in progress. This is only a ”node” update that we are aware it is required and does not impact the payload itself, which was the scope of this review.

R0bert: Acknowledged.

3.2.3 Deployer signer/key mismatch risk

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The deployment flow currently relies on two independent operator inputs that are expected to represent the same signer identity, but this relationship is not enforced in code. The deployment script reads a `DEPLOYER` address from environment variables and starts broadcast with `vm.startBroadcast(deployer)`, while the wrapper script separately requires and passes `ROLLUP_DEPLOYMENT_PRIVATE_KEY` to Forge. Because there is no explicit check that the provided address

equals the address derived from the provided private key, an operator can supply mismatched values and still proceed into an execution attempt.

This creates an avoidable operational hazard during the upgrade. If the requested sender and signing key do not line up the deployment will revert.

Recommendation: Consider making the signer source unambiguous and enforce identity consistency. Prefer a single source of truth by broadcasting from the private key directly, or add an explicit assertion that `DEPLOYER` matches `vm.addr(privateKey)` before calling `vm.startBroadcast`.

```
address deployer = vm.envAddress("DEPLOYER");
uint256 pk = vm.envUint("ROLLUP_DEPLOYMENT_PRIVATE_KEY");
require(deployer == vm.addr(pk), "DEPLOYER != key-derived signer");
vm.startBroadcast(pk);
```

Aztec Labs: Acknowledged. As mentioned it will revert as Foundry should be throwing an error if the key matching the `DEPLOYER` is not available. So if you have different values for the two, and are using `--broadcast` foundry will give you an error for it, and that seems fine with me. I would keep as is because I like the clarity of the broadcast with specific address, and also not having the key loaded directly into the script code. Essentially a style choice.

R0bert: Acknowledged.

3.2.4 1/2/3 slash votes all apply same penalty

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The tally slashing flow supports three severity levels through encoded vote units. Proposers submit vote data where each validator can receive 0, 1, 2, or 3 units. During execution, the contract tallies those units, checks quorum and then maps the winning unit level to one of three penalty constants: `SLASH_AMOUNT_SMALL`, `SLASH_AMOUNT_MEDIUM` or `SLASH_AMOUNT_LARGE`. The mechanism is therefore designed to let higher-severity consensus produce a larger slash.

```
uint256 internal constant AZTEC_SLASH_AMOUNT_SMALL = 2000e18;
uint256 internal constant AZTEC_SLASH_AMOUNT_MEDIUM = 2000e18;
uint256 internal constant AZTEC_SLASH_AMOUNT_LARGE = 2000e18;
```

In the reviewed configuration, all three penalty constants are equal. This means the protocol still accepts 1-unit, 2-unit, and 3-unit votes, but the final slash amount is identical in every case once quorum is met. The severity labels remain in the flow, but they no longer change economics.

```
if (j == 1) {
  slashAmount = SLASH_AMOUNT_SMALL;
} else if (j == 2) {
  slashAmount = SLASH_AMOUNT_MEDIUM;
} else if (j == 3) {
  slashAmount = SLASH_AMOUNT_LARGE;
}
```

A concrete example clarifies the effect. If quorum is reached at 1 unit, the validator is slashed by `2000e18`. If quorum is reached at 2 units, the validator is still slashed by `2000e18`. If quorum is reached at 3 units, the validator is again slashed by `2000e18`. Different severity outcomes therefore converge to the same penalty.

It is also important to mention that the flat-tier condition is not newly introduced by the alpha cutover constants. A direct `cast` read on the current canonical mainnet slashing proposer before upgrade already returns the same values for `SLASH_AMOUNT_SMALL`, `SLASH_AMOUNT_MEDIUM` and `SLASH_AMOUNT_LARGE`, each equal to `2000e18`. The upgrade therefore carries this policy forward rather than creating a fresh regression.

This collapses the intended severity ladder and removes meaningful differentiation between lower-severity and higher-severity slash signaling.

Recommendation: Configure strictly increasing slash tiers so vote severity maps to distinct penalties. If equal penalties are intentional for a temporary phase, document that decision explicitly and add a scheduled governance follow-up to restore differentiated levels.

Aztec Labs: Acknowledged. Currently, this configuration is intended.

R0bert: Acknowledged.