Aztec PR 599

Security Review

Cantina Solo review by:
R0bert, Lead Security Researcher

December 17, 2025

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 Nov 10th to Nov 11th the security researchers conducted a review of ignition-monorepo on commit hash 2a2c7909. A total of **4** issues were identified:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	1	0	1
Gas Optimizations	1	1	0
Informational	2	0	2
Total	**4**	**1**	**3**

2.1 Scope

The security review had the following components in scope for ignition-monorepo on commit hash 2a2c7909:

```
contracts/src/ProtocolTreasury.sol
```

(Files changed on PR 599)

3 Findings

3.1 Low Risk

3.1.1 Governance could bypass insider gate by lowering ATP registry timestamp when colluding with registry owner

Severity: Low Risk

Context: ProtocolTreasury.sol#L118-L121

Description: `relay()` recomputes `insiderCanActTimestamp()` every call by reading `ATP_REGISTRY.getExecuteAllowedAt()` and adding seven days. The ATP registry owner, however, can decrease `executeAllowedAt` arbitrarily via `Registry.setExecuteAllowedAt`. If that owner cooperates with governance (or is compromised by it), they can:

1. Lower `executeAllowedAt` for a single transaction so that `insiderCanActTimestamp()` is satisfied immediately.
2. Execute `relay()` to drain funds or push approvals before insiders were genuinely allowed to act.

This completely defeats the stated guarantee that insiders receive at least a week of participation time before governance can use the relayer. Any trust in the gate devolves into ”trust the registry owner not to cheat,” which is the very scenario the relayer was meant to prevent.

Recent design notes indicate the insider ATP registry is operated by a separate entity rather than governance. That setup prevents unilateral governance bypasses, but it still leaves a single mutable timestamp whose guardian can (intentionally or accidentally) invalidate the delay guarantees. If governance colludes with, compromises, or exerts pressure on that operator, the same three-step attack above becomes possible. Lowering `executeAllowedAt` also shortens insiders' ”time-to-act”, so the timestamp remains a powerful lever that should not be mutable once insiders have deposited funds.

Recommendation: Decouple the relayer from future registry mutations. Either (a) pass the final unlock timestamp at deployment and store it immutably (adding the 7-day buffer internally), or (b) cache the first-observed `executeAllowedAt` and refuse to ever decrease the cached value, optionally only allowing increases. Additionally, consider transferring registry ownership to an independent contract that cannot collude with governance before insiders unlock.

Aztec Labs: Acknowledged. The ATP registry owner (of the one where insider funds are) is not the governance, but a separate entity. Collusion could still create the issue though, but less likely. Moreover, it is not possible to restore the `executeAllowedAt` since it can only be decreased, e.g., if it is lowered it will stay low. Anyway, they could still allow the execution of a already queued proposal to execute as if it was possible for ATP's following that registry to act (without actually haven been able to).

However, the insiders (ATP's following registry) have significant trust assumptions towards the registry owner already. As he will need to add an implementation that they can use to participate in governance for the period after `executeAllowedAt` until the assets are liquid. So if compromised, he could avoid them participating anyway by simply not adding a way to deposit.

In short, it seems fine to me that we rely on the trust assumption for honest `executeAllowedAt` values, since we already rely on it being honest for more powerful actions.

R0bert: Acknowledged.

3.2 Gas Optimization

3.2.1 Zero-value relays pay unnecessary BALANCE gas

Severity: Gas Optimization

Context: ProtocolTreasury.sol#L123-L124

Description: `relay` always executes

```
require(address(this).balance >= value, Errors.InsufficientBalance(...))
```

Even when `value == 0`, the contract performs a `BALANCE` opcode and allocates memory for the revert data, which costs ~400 gas per governance action, the common case if most relays don't transfer native assets. Since a zero transfer can never underflow the relayer's balance, the check can safely be skipped in that branch; alternatively, rely on the target call reverting naturally if balance is insufficient.

Recommendation: Wrap the balance check in `if (value != 0)` before executing it, so zero-value relays bypass the BALANCE opcode and associated memory work. This yields a small but recurring gas savings for governance executions that only call functions without native asset transfers.

Aztec Labs: Fixed in PR 690.

R0bert: Fix verified.

3.3 Informational

3.3.1 ProtocolTreasury inefective bootstrap on mature governance

Severity: Informational

Context: ProtocolTreasury.sol#L46

Description: `markNext()` only processes a single proposal per transaction and enforces that the proposal is already in a terminal state before incrementing `markedProposalsCount`. Because the counter is initialized to zero, deploying the relayer against an existing governance with N historical proposals requires N separate transactions, one for each proposal, to advance the pointer, since each invocation touches storage and the contract provides no batching primitive. For a live system that has already created thousands of proposals, this means thousands of keeper calls (e.g., 10000 proposals $\rightarrow$ 10000 distinct `markNext` transactions) must be executed before the relayer can even consider forwarding a call. Until the backlog is cleared, `relay` always sees an unmarked, pre-insider proposal and reverts on the insider-check. Impact: onboarding `ProtocolTreasury` to a mature governance can entail a prohibitive amount of operational work and may be infeasible in practice if keepers cannot reliably submit that many transactions.

Recommendation: Add an initialization override so a trusted role can set `markedProposalsCount`, or extend `markNext` with a batched variant (`markRange(uint256 count)`) that iterates in-contract and emits events for the processed proposals while staying within block gas limits.

Aztec Labs: Acknowledged. It is already possible to do a batch by using a multicall or similar, so we don't think that we need a `markRange` (though it was considered at first, but left out for simplicity). We don't think it is too much a concern in our setup because it is deployed along with the governance, so there are existing proposals and also how new proposals are created in our system. Adding some context, a governance proposal can be made in 2 ways:

1. Locking a large amount for a significant time period (~2% of supply and locked for 90 days).
2. Going through the ”governance proposer” which is based on validator actions and at most one proposal can be made every 20 hours.

So even a ”spam” of proposals would grow very slow here and should have a higher cost to create than consume. Agree for a mature governance though that if there are a lot of proposals, but even then I think it should be no big hurdle.

R0bert: Acknowledged.

3.3.2 Relay strips governance caller identity

Severity: Informational

Context: ProtocolTreasury.sol#L125

Description: `relay` executes targets via a standard `call` originating from the relayer contract. Any contract that used to rely on `msg.sender == GOVERNANCE` will now see the `ProtocolTreasury` address instead. Once ownership or privileged roles are transferred to the relayer, the governance account no longer has those rights; the relayer becomes the effective admin. If downstream contracts still gate logic on the governance address and cannot be reconfigured to trust the relayer, those privileges are effectively lost when relaying.

Recommendation: Before routing calls through `ProtocolTreasury`, ensure all target contracts explicitly trust the `ProtocolTreasury`'s address (e.g., by transferring ownership to it or adding it to ACLs).

Aztec Labs: Acknowledged.

R0bert: Acknowledged.