CANTINA

Aztec Ignition

Security Review

Cantina Managed review by:

Desmond Ho, Lead Security Researcher
Hake, Associate Security Researcher

September 10, 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

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 Aug 22nd to Aug 29th the Cantina team conducted a review of ignition-monorepo on commit hash c9efabac. The team identified a total of **11** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	2	1	1
Gas Optimizations	4	3	1
Informational	4	2	2
Total	**11**	**7**	**4**

3 Findings

3.1 Medium Risk

3.1.1 Provider can atomically increase it's take rate

Severity: Medium Risk

Context: StakingRegistry.sol#L371

Summary: The `updateProviderTakeRate()` function allows provider administrators to modify their take rate immediately without any timelock or notification mechanism. This enables providers to unexpectedly increase their fees, potentially reducing user rewards below expected levels after users have already committed to staking.

Finding Description: The StakingRegistry contract allows provider administrators to update their take rate through the `updateProviderTakeRate()` function. The function performs basic validation to ensure the caller is the provider admin and that the new rate doesn't exceed the maximum allowed value (`Constants.BIPS`). However, there are no protective mechanisms to prevent sudden changes that could adversely affect users.

When users stake with a provider, they likely make their decision based on the current take rate. If a provider can change this rate instantly after users have staked, users may end up receiving significantly lower rewards than anticipated. This creates an asymmetric risk where users commit their funds based on one set of terms, but those terms can be altered unilaterally by the provider.

Impact Explanation: The impact is medium as users may receive lower staking rewards than expected if a provider increases their take rate after users have staked. While this doesn't result in direct loss of principal funds, it does affect the economic assumptions users made when choosing to stake, potentially resulting in opportunity costs and reduced yields.

Likelihood Explanation: The likelihood is medium. While most providers would likely act in good faith to maintain their reputation, the lack of technical constraints would enable this to easily happen even if by accident. A provider only needs to call a single function to immediately change rates, requiring no special conditions or complex setup.

Recommendation: Consider implementing a timelock mechanism for `providerTakeRate` changes to protect users and maintain trust in the system.

Aztec Labs: Fixed in commit 195a455f.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Admin role transfer lacks validation and acceptance mechanism

Severity: Low Risk

Context: StakingRegistry.sol#L331-L336

Description: The `updateProviderAdmin()` function in the StakingRegistry contract allows provider admins to transfer their role to a new address without requiring the new admin to accept the role. Consequently, admin privileges could be accidentally transferred to an incorrect or uncontrolled address. Additionally, the function does not validate that the new admin address differs from the current admin address, potentially allowing unnecessary state changes.

Recommendation: Consider implementing a two-step admin transfer process where the new admin must explicitly accept the role. Additionally, validate that the new admin differs from the current admin:

```
+ mapping(uint256 => address) public pendingProviderAdmins;

  function updateProviderAdmin(uint256 _providerIdentifier, address _newAdmin) external
  ↪ override(IStakingRegistry) {
      require(
          msg.sender == providerConfigurations[_providerIdentifier].providerAdmin,
          StakingRegistry__NotProviderAdmin()
      );
      require(_newAdmin != address(0), StakingRegistry__ZeroAddress());
```

```
+     require(_newAdmin != providerConfigurations[_providerIdentifier].providerAdmin, "Same admin address");
-     providerConfigurations[_providerIdentifier].providerAdmin = _newAdmin;
-     emit ProviderAdminUpdated(_providerIdentifier, _newAdmin);
+     pendingProviderAdmins[_providerIdentifier] = _newAdmin;
+     emit ProviderAdminTransferInitiated(_providerIdentifier, _newAdmin);
  }

+ function acceptProviderAdmin(uint256 _providerIdentifier) external {
+     require(msg.sender == pendingProviderAdmins[_providerIdentifier], "Not pending admin");
+     providerConfigurations[_providerIdentifier].providerAdmin = msg.sender;
+     delete pendingProviderAdmins[_providerIdentifier];
+     emit ProviderAdminUpdated(_providerIdentifier, msg.sender);
+ }
```

Aztec Labs: Fixed in commit 931a1b73.

Cantina Managed: Fix verified.

3.2.2 Duplicate `KeyStore`s can be added causing rollup deposit failures

Severity: Low Risk

Context: StakingRegistry.sol#L293

Description: The `addKeyToProvider()` and `addKeysToProvider()` function in the StakingRegistry contract does not validate whether a keyStore already exists before adding it to the provider queue. When a duplicate `keyStore` is added to the queue and subsequently processed for rollup deposits, the second deposit attempt with the same key will fail. This creates an operational issue where the protocol relies on the operator's diligence to avoid providing duplicate keys rather than enforcing this constraint at the contract level. While this does not directly result in fund loss, it can lead to operational delays and increased gas costs from failed transactions.

Recommendation: Consider implementing a duplicate check mechanism before adding keystores to the provider queue. One approach could be maintaining a mapping to track existing `keyStore`s.

Aztec Labs: Acknowledged. We cannot prevent this issue from all angles, since there will be keys going into the system which are not from this staking registry. We will gather more information on how to address this past what was recommended.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 Inefficient mechanism for removing multiple keys from provider queue

Severity: Gas Optimization

Context: StakingRegistry.sol#L391

Description: The `dripProviderQueue()` function in the StakingRegistry contract allows provider admins to remove keys from their queue one at a time. This design becomes inefficient when multiple keys need to be removed. Each key removal requires a separate transaction, leading to increased gas costs and operational overhead.

Recommendation: Consider implementing a batch removal function that allows provider admins to remove multiple keys in a single transaction. This approach would significantly reduce gas costs and improve operational efficiency when managing provider queues.

Aztec Labs: Fixed in commit de631388.

Cantina Managed: Fix verified.

3.3.2 Loop initialization can be optimized by using default zero value

Severity: Gas Optimization

Context: StakingRegistry.sol#L314

Description: The loop initialization in the contract explicitly initializes the counter variable to zero (`uint256 i = 0`), which is unnecessary since `uint256` variables are zero-initialized by default in Solidity.

Recommendation: Consider using the default zero initialization to save a small amount of gas:

```
- for (uint256 i = 0; i < _keyStores.length; i++) {
+ for (uint256 i; i < _keyStores.length; i++) {
```

Aztec Labs: Fixed in commit 23913401.

Cantina Managed: Fix verified.

3.3.3 `finaliseWithdrawal()` can be permissionless

Severity: Gas Optimization

Context: ATPWithdrawableStaker.sol#L52

Description: `finaliseWithdrawal()` doesn't need the `onlyOperator` modifier since it's permissionless on the rollup.

Recommendation:

```
- onlyOperator
```

Aztec Labs: Fixed in commit 0ec876d5.

Cantina Managed: Fix verified.

3.3.4 `enqueue()` has redundant return value

Severity: Gas Optimization

Context: QueueLib.sol#L20

Description: The return value of `enqueue()` is unused in all its invocations.

Recommendation: Consider removing the return value.

Aztec Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Inconsistent increment syntax in queue operations

Severity: Informational

Context: QueueLib.sol#L33

Description: The queue implementation uses inconsistent syntax for incrementing position counters. In the `enqueue()` function, the last position is incremented using `queueLocation + 1`, while in the `dequeue()` function, the first position is incremented using the += operator. This inconsistency reduces code readability and maintainability.

Recommendation: Consider standardizing the increment syntax across both functions for better consistency. The most concise approach would be to use the increment (++) operator (although not necessarily the most gas efficient).

Aztec Labs: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 Key addition management code quality improvements

Severity: Informational

Context: StakingRegistry.sol#L285

Description/Recommendation:

1. The `addKeyToProvider()` function is unnecessary since `addKeysToProvider()` can handle both single and multiple key additions. Consider removing `addKeyToProvider()`.

2. The `addKeysToProvider()` function should require `_keyStores.length > 0` to prevent adding empty keys to the queue and wasting gas.
3. The `addKeysToProvider()` function contains no checks to validate whether `_keyStores` values are populated. Consider checking `_keyStores[i].attester != address(0)`.

Aztec Labs: Fixed in commit c95aacd7.

Cantina Managed: Fix verified.

3.4.3 Typographical Errors

Severity: Informational

Context: ATPWithdrawableStaker.sol#L27, ATPWithdrawableStaker.sol#L47, BN254.sol#L18, StakingRegistry.sol#L145

Description/Recommendation:

```
- which is set the the ATP address
+ which is set as the ATP address


- require to be called via the staker
+ required to be called via the staker


- pairied
+ paired


- Retreive
+ Retrieve
```

Aztec Labs: Fixed in commit 20acd71f.

Cantina Managed: Fix verified.

3.4.4 Slashing induced exits enables earlier withdrawals

Severity: Informational

Context: ATPWithdrawableStaker.sol#L49

Description: It might be beneficial to be slashed to be able to withdraw and trade earlier than others, since `initiateWithdraw()` doesn't update the exit delay timer for slashing induced exits. This depends on the value of the slash amount vs value of being able to sell the withdrawable amount after slashing at an earlier time.

Recommendation: Since the slash amount is variable, consider imposing heavier slash penalties in the network's infancy stage.

Aztec Labs: Acknowledged.

Cantina Managed: Acknowledged.