Atlas v1.6
Security Review

Cantina Managed review by:

Riley Holterhus, Lead Security Researcher
Blockdev, Security Researcher

May 9, 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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

Atlas is a generalized Execution Abstraction protocol developed by FastLane Labs that aims to reduce the complexity and cost associated with deploying application-specific order flow auctions (OFAs) while optimizing transaction processing and improving value capture within blockchain ecosystems.

From Apr 28th to May 1st the Cantina team conducted a review of atlas on commit hash 499b044d. The team identified a total of **9** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	2	2	0
Gas Optimizations	1	1	0
Informational	4	3	1
Total	**9**	**8**	**1**

3 Findings

3.1 Medium Risk

3.1.1 Solvers may receive less gas than they expect

Severity: Medium Risk

Context: Escrow.sol#L367-L368

Description: A key change introduced in Atlas V1.6 is that the gas limit forwarded to solvers is now set as the minimum of `solverOp.gas` (the value set by the solver) and `dConfig.solverGasLimit` (the value from the `DAppControl`).

One consequence of this change is that if `dConfig.solverGasLimit` is lowered after a solver signs and submits their solverOp, the gas limit they ultimately receive will be lower than they expect. In a worst-case scenario, a `DAppControl` could intentionally lower the `dConfig.solverGasLimit` value to cause the solverOp to fail due to an out-of-gas error. This could grief the solver since an out-of-gas error would be treated as their fault.

Recommendation: Consider adding a field to the userOp struct to commit to the expected `dConfig.solverGasLimit` from the DAppControl, similar to how `bundlerSurchargeRate` is included in the userOp. This approach would only add one new field and ensures that all solverOps are aware of the expected `dConfig.solverGasLimit`, since it contributes to the userOp hash.

Fastlane: Fixed in commit 1ef75ad and commit 8b217b3.

Cantina Managed: Verified.

3.1.2 Net repayments can be counted across multiple `solverOps`

Severity: Medium Risk

Context: GasAccounting.sol#L605

Description: A `solverOp`'s balance is considered reconciled if all current repayments are at least equal to all current borrows, and the solver has also prepaid their maximum gas liability, either through approving their bonded atlETH or with an excess repayment. This is implemented in the following code:

```
function _isBalanceReconciled() internal view returns (bool) {
    // ...
    return (bL.repays >= bL.borrows) && (_maxApprovedGasValue + _netRepayments >= gL.solverGasLiability());
}
```

However, this logic does not function correctly when `multipleSuccessfulSolvers == true`, because the same net repayment can be counted toward multiple solverOps.

For example, if one `solverOp` leaves behind a 1 ETH net repayment and there are 10 `solverOps`, each can independently apply the same 1 ETH toward their own gas liability check. There is no mechanism to track how much of the repayment each `solverOp` is relying on, which allows the same ETH to be reused across multiple `solverOps`. As a result, `_isBalanceReconciled()` may incorrectly return true for all `solverOps`, even if the excess is only sufficient to cover one.

This can lead to a shortfall in the gas reimbursement that only becomes apparent during the final `_settle()` call, at which point the bundler would implicitly cover the difference.

Recommendation: Change the gas accounting logic so that after each solverOp, the portion of the net repayment used toward its gas liability is subtracted out, preventing other solverOps from reusing the same amount.

Fastlane: Fixed in PR 482 and PR 483.

Cantina Managed: Verified. The chosen fix is to exclude net repayments from the gas liability check when `multipleSuccessfulSolvers == true`, which also fixes the issue.

3.2 Low Risk

3.2.1 `solverGasLiability()` overestimated when `multipleSuccessfulSolvers == true`

Severity: Low Risk

Context: GasAccLib.sol#L71-L82

Description: The `solverGasLiability()` function estimates the upper bound amount of gas a successful solver must pay. Its implementation predates the `multipleSuccessfulSolvers == true` configuration option and assumes a successful solver must cover all overhead gas costs (e.g. userOp and dapp hook costs). However, when `multipleSuccessfulSolvers` is true, solvers only need to pay for their own calldata and execution gas costs. As a result, the current implementation overestimates the gas liability and requires larger prepayments than necessary.

Recommendation: Consider updating the accounting logic so that when `multipleSuccessfulSolvers == true`, successful solvers only need prepay their own execution and calldata costs.

Fastlane: Fixed in PR 480 by changing the `_initialRemainingMaxGas` value set at the start of a metacall.

Cantina Managed: Verified.

3.2.2 Unsafe typecasts

Severity: Low Risk

Context: Atlas.sol#L297, GasAccounting.sol#L56-L59, GasAccounting.sol#L163, GasAccounting.sol#L303, GasAccounting.sol#L313, GasAccounting.sol#L345, GasAccounting.sol#L350, GasAccounting.sol#L363, ChainlinkAtlasWrapper.sol#L90

Description: Several variables are downcasted from `uint256` to `uint40` and there are no guarantees for some of them to not overflow. Hence, it's better to use `SafeCast`.

Recommendation: Use `SafeCast` to revert if variables overflow `uint40`.

Fastlane: Fixed in commit 5728314 and commit fb9eac9.

Cantina Managed: Verified.

3.3 Gas Optimization

3.3.1 Use OZ `Math.min()`

Severity: Gas Optimization

Context: Atlas.sol#L295-L296, GasAccounting.sol#L300

Description: OZ's `Math.min()` can be used instead of the ternary operator to compute a minimum of two values. It's easier to read and may also save some gas as `Math.min()` avoids JUMPs.

Recommendation: Use `Math.min()`.

Fastlane: Fixed in commit 482c43b. It's also relevant to `Escrow.sol`, `AtlasVerification.sol`, `Simulator.sol`, and `Sorter.sol`.

Cantina Managed: Verified.

3.4 Informational

3.4.1 Unused surcharge mask constants

Severity: Informational

Context: AtlasConstants.sol#L49-L52

Description: The `_ATLAS_SURCHARGE_MASK` and `_BUNDLER_SURCHARGE_MASK` constants in `AtlasConstants` are unused. These variables are no longer relevant because the bundler surcharge is now a configurable value in the DAppControl instead of being stored in Atlas, and the Atlas surcharge isn't in a packed storage variable.

Recommendation: Consider removing the unused `_ATLAS_SURCHARGE_MASK` and `_BUNDLER_SURCHARGE_MASK` variables.

Fastlane: Fixed in commit b036575.

Cantina Managed: Verified.

3.4.2 Incorrect comments in `_verifyCallConfig()`

Severity: Informational

Context: AtlasVerification.sol#L285-L296

Description: The `multipleSuccessfulSolvers` configuration option is not allowed when `allowsZeroSolvers`, `allowsSolverAuctioneer`, or `needsFulfillment` are true. While this logic is correctly enforced in the code, the related comments for these three options incorrectly reference `invertsBidValue` instead.

Recommendation: Update the comments to reference the correct disallowed options.

Fastlane: Fixed in commit 5613618.

Cantina Managed: Verified.

3.4.3 Analytics consider all solvers as failed when `multipleSuccessfulSolvers == true`

Severity: Informational

Context: GasAccounting.sol#L328

Description: When `multipleSuccessfulSolvers` is true, the gas accounting is handled as if all solverOps reverted. This can be seen by the fact that `_executeSolverOperation()` always calls `_handleSolverFailAccounting()` when `multipleSuccessfulSolvers == true`, regardless of whether the solverOp succeeded or failed.

One side effect of this is that the analytics are also updated as if every solver failed, even though some of the solvers may have succeeded and some of them may have failed.

Recommendation: Consider updating the logic so that when `multipleSuccessfulSolvers` is `true`, the analytics correctly distinguish between successful and failed solvers.

Fastlane: Acknowledged.

Cantina Managed: Acknowledged.

3.4.4 Missing Natspec

Severity: Informational

Context: GasAccounting.sol#L439-L449

Description: A new argument `multipleSuccessfulSolvers` is added to `_settle()` but it's not added to its Natspec.

Recommendation: Add Natspec for `multipleSuccessfulSolvers`.

Fastlane: Fixed in commit 9dbafd2.

Cantina Managed: Verified.