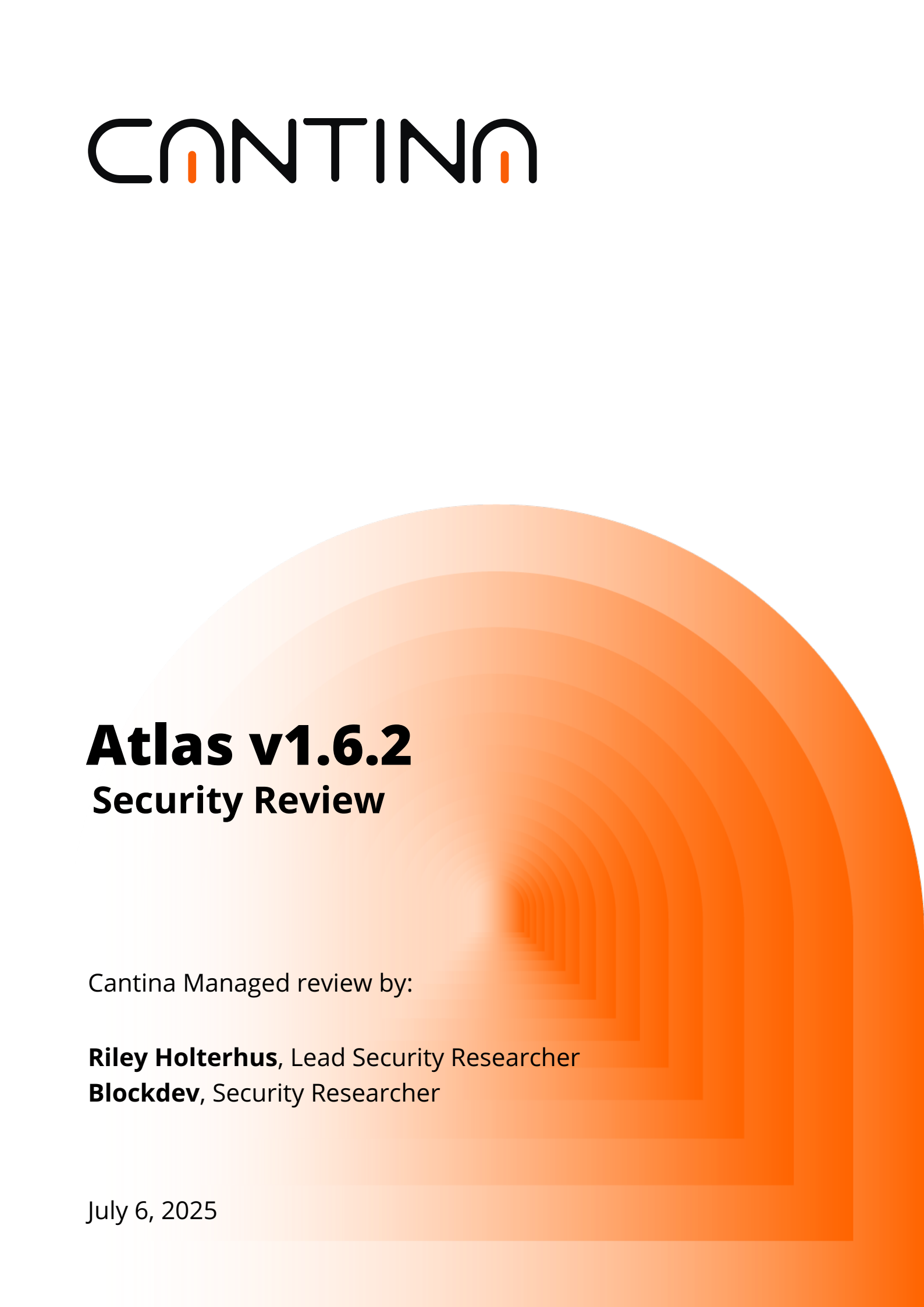

Atlas v1.6.2
Security Review

Cantina Managed review by:

Riley Holterhus, Lead Security Researcher
Blockdev, Security Researcher

July 6, 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 Jun 27th to Jun 30th the Cantina team conducted a review of Atlas-v1.6.2 on commit hash 1eda7c6e. The team identified a total of **6** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	1	1
Gas Optimizations	1	1	0
Informational	3	1	2
Total	**6**	**3**	**3**

3 Findings

3.1 Low Risk

3.1.1 `_estimateMetacallGasLimit()` excludes some Atlas gas overhead

Severity: Low Risk

Context: Simulator.sol#L66-L102

Description: In the `Simulator` contract, the `_estimateMetacallGasLimit()` function is meant to estimate the total gas cost (including calldata and execution gas) for calling the main `metacall()` function with a given `userOp` and `solverOps` array.

The function accounts for all external calls made during a `metacall()` (e.g. calls to the execution environment, solvers, and DappControl). However, one area it underestimates is the gas cost of the Atlas code itself.

Currently, the only two sections of the Atlas overhead code that it adds to its calculations are the `_FIXED_GAS_OFFSET` (which represents the gas cost after the last `gasleft()` checkpoint in the `metacall()`), and the `_BID_FIND_OVERHEAD` (which represents the approximate overhead each solverOp uses for sorting when `exPostBids` is true). This means that other sections of the Atlas code, such as `validateCalls()` and each of the `_executeSolverOperation()` calls, are not included in the gas estimate returned by `_estimateMetacallGasLimit()`.

Recommendation: Consider adding logic to estimate an upper bound for some of the main sections of the Atlas code that are currently not included in the `_estimateMetacallGasLimit()` function.

Fastlane: Addressed in PR 493 This PR had the following changes:

- Added `_PRE_EXECUTE_METACALL_GAS` constant of 150k gas.
- Renamed `_FIXED_GAS_OFFSET` to `_POST_SETTLE_METACALL_GAS` and lowered it to 70k gas.
- Added an `_EXECUTE_SOLVER_OVERHEAD` constant of 45k gas which represents an upper estimate of the gas used in `_executeSolverOperation()`, excluding the `_solverOpWrapper()` call.

Cantina Managed: Verified.

3.1.2 `IL2GasCalculator` doesn't consider OP stack operator fees

Severity: Low Risk

Context: BaseGasCalculator.sol#L1-L70

Description: To account for gas cost differences across various L2 chains, the `Atlas` contract supports an optional `L2_GAS_CALCULATOR`, which implements the `IL2GasCalculator` interface. When set, its `getCalldataGas()` and `initialGasUsed()` functions are used to compute L2-specific gas costs.

Note that in May 2025 the Isthmus upgrade was introduced to the OP stack, which added a new optional fee mechanism known as the operator fee. This fee consists of two parts: the `operatorFeeConstant` (a fixed fee per transaction) and the `operatorFeeScalar` (a fee per unit of gas similar to the base fee and priority fee).

Since the two functions currently in the `IL2GasCalculator` interface only relate to calldata gas, there doesn't seem to be a way to account for either component of the operator fee in an `L2_GAS_CALCULATOR` implementation. While the operator values are currently set to 0 on many OP stack chains such as Optimism, Base, and Unichain, this may lead to discrepancies in the future if they are non-zero.

Recommendation: Consider extending the `IL2GasCalculator` interface to support the new OP stack operator fee. Note that this may require a large refactor of some of the gas accounting code.

Fastlane: Acknowledged. No fix yet as it would require a larger refactor to support Operator Fee on OP Stack L2s, and the risk seems to just be that the bundler will be slightly under-compensated by solvers, which is acceptable as FastLane will be the bundler in all currently planned deployments.

Also note that a possible work around for the `operatorFeeScalar` is increasing the bundler surcharge rate which would require no code changes.

Added comments documenting these ideas in commit 283c505 and commit 096be95.

Cantina Managed: Acknowledged.

3.2 Gas Optimization

3.2.1 Bump `optimizer_runs`

Severity: Gas Optimization

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

Description: `optimizer_runs` is set to `8` currently. It's intentionally set to a low value to avoid triggering the contract size limit imposed by EVM chains. However, it is underestimated and can be increased to a higher value.

Recommendation: Bump `optimizer_runs` to an appropriately higher value.

Fastlane: Fixed in commit 1da97b8.

Cantina Managed: Verified.

3.3 Informational

3.3.1 Simulator calls `metacall()` with `_BASE_TX_GAS_USED` already paid

Severity: Informational

Context: Simulator.sol#L313-L319

Description: In the `Simulator` contract, the `metacallSimulation()` function only forwards the estimated execution gas needed for the `metacall()` and does not include calldata gas. This is because the call is being made within a contract already, so the calldata-related gas costs have already been paid earlier in the simulation flow. This is described in the following comments:

```
function metacallSimulation(/* ... */) /* ... */ {
    // ...
    // In real Atlas metacalls, when the caller is an EOA, it should include the suggested calldata gas in the
    ↪  gas
    // limit. However, in as this is a Simulator call, that calldata gas has already been deducted in the
    ↪  initial
    // `simUserOperation()` or `simSolverCall()` call. As such, we set the metacall gas limit here to just the
    // suggested execution gas.

    bool auctionWon =
        IAtlas(atlas).metacall{ value: msg.value, gas: metacallExecutionGas }(userOp, solverOps, dAppOp,
        ↪  address(0));

    // ...
}
```

It's also worth noting that the `_BASE_TX_GAS_USED == 21_000` gas required for initiating a transaction has also already been consumed by the outer contract call. This means the `metacall()` itself receives `21_000` more gas than would be available in a real transaction.

Recommendation: Consider adjusting the simulation to account for the fact that `_BASE_TX_GAS_USED` has already been paid before `metacall()` is called. Alternatively, since the `Simulator` is not meant to be 100% perfect with gas estimation, consider documenting this behavior.

Fastlane: Fixed in commit c37ad05.

Cantina Managed: Verified. The `_BASE_TX_GAS_USED` is now subtracted from the execution gas amount forwarded to `metacallSimulation()`.

3.3.2 `_GAS_PER_CALLDATA_BYTE` considerations with EIP-7623

Severity: Informational

Context: GasAccLib.sol#L37-L39

Description: In the Atlas codebase, `_GAS_PER_CALLDATA_BYTE` is the gas charge per byte of calldata used when estimating calldata gas. It is currently set to 8, which is half the cost of a non-zero byte (16 gas)

and double the cost of a zero byte (4 gas) under the current Ethereum calldata costs. It's worth noting that EIP-7623 introduces additional logic for calldata costs. For any transaction, the network will also now compute:

```
calldata_floor = (10 x num_zero_bytes) + (40 x num_nonzero_bytes)
```

And if `calldata_floor` exceeds the gas the transaction would normally consume (execution gas plus legacy calldata gas), the transaction is charged `calldata_floor` instead. This new formula essentially sets a lower bound on total gas that depends only on the calldata, and execution gas is ignored in this new calculation.

Since Atlas transactions will be mainly execution costs, it's very unlikely that this new calldata pricing will ever be used for an Atlas transaction, so keeping the `_GAS_PER_CALLDATA_BYTE` as is seems appropriate.

Recommendation: No code changes seem necessary, and this finding has been provided for informational purposes. Consider documenting this in the code.

Fastlane: Acknowledged. We agree the new calldata gas model shouldn't ever affect Atlas due to how large execution gas is in proportion to calldata gas in all planned scenarios.

Cantina Managed: Acknowledged.

3.3.3 `_errorCatcher()` has hidden gas costs before reaching `metacall()`

Severity: Informational

Context: Simulator.sol#L252-L265

Description: The `_errorCatcher()` function in the `Simulator` contract calls `metacallSimulation()`, which then calls the main `metacall()` function. Since this involves two external calls before reaching `metacall()`, each one forwards at most 63/64 of the remaining gas. To account for this, `_errorCatcher()` calculates a `minGasLeft` value based on the expected `metacallExecutionGas`, scaled up by $\left(\frac{64}{63}\right)^2$, and adds a buffer:

```
function _errorCatcher(/* ... */) /* ... */ {
    // ...
    // For each external call, a max of 63/64 of the gas left is forwarded.
    // Therefore we should have at least [metacallExecutionGas * (64/63) ^ 2] at this point to ensure there
    ↪  will be
    // enough gas at the start of the `metacall()`
    uint256 minGasLeft = metacallExecutionGas * _MIN_GAS_SCALING_FACTOR / _SCALE + _ERROR_CATCHER_GAS_BUFFER;
    uint256 gasLeft = gasleft();

    if (gasLeft < minGasLeft) {
        revert InsufficientGasForMetacallSimulation(
            gasLeft, // The gas left at this point that was insufficient to pass the check
            metacallExecutionGas + metacallCalldataGas, // Suggested gas limit for a real metacall
            _SIM_ENTRYPOINT_GAS_BUFFER + minGasLeft + metacallCalldataGas // Suggested gas limit for a sim call
        );
    }
    // ...
}
```

This check is meant to ensure that by the time `metacall()` is reached, the necessary execution gas is still available.

However, right after this check, Solidity may use some gas to allocate memory and prepare the external call. The same applies inside `metacallSimulation()` before calling `metacall()`. As a result, even if the `gasLeft` check passes, slightly less gas than expected may reach the actual `metacall()`.

Recommendation: Consider whether this needs to be accounted for. If not, consider documenting this behavior for clarity. If yes, increasing the `_ERROR_CATCHER_GAS_BUFFER` slightly could help cover the extra memory setup cost.

Fastlane: Acknowledged. It is recommended that auctioneers/bundlers call these Simulator view functions with really large gas limits, as only the necessary gas limit will actually be forwarded to Atlas to simulate the metacall. Added a note in the code advising that in commit cfa3e4d.

Cantina Managed: Acknowledged.