Byzantine Finance Atlas.sol

Security Review

Cantina Managed review by:
Rikard Hjort, Lead Security Researcher
Om Parikh, Security Researcher

December 15, 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

Byzantine is a trustless and efficient restaking layer with permissionless strategy creation.

From Dec 6th to Dec 7th the Cantina team conducted a review of batch-call-and-sponsor on commit hash 8630b9e8. The team identified a total of **6** issues:

Issues Found

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

The Cantina Managed team reviewed Byzantine Finance's batch-call-and-sponsor holistically on commit hash 66ce52a9 and concluded that all findings were addressed and no new vulnerabilities were identified.

2.1 Scope

The security review had the following components in scope for batch-call-and-sponsor on commit hash 8630b9e8:

```
src/Atlas.sol
```

3 Findings

3.1 High Risk

3.1.1 Missing `receive` / `fallback`, ERC721 & ERC1155 token accepting hooks

Severity: High Risk

Context: Atlas.sol#L142

Description: `Atlas.sol` contract doesn't have a `receive` or `fallback` function, so once the delegation is active, user will not be accept ether during `execute` and call / batch will revert. Similarly, the Altas delegation cannot receive `ERC721` and `ERC1155` tokens as `onERC721Received` and `onERC1155Received` / `onERC1155BatchReceived` are missing.

Proof of Concept:

- Add this ether sender contract:

```
contract SomeSenderContract {
    receive() external payable {}

    function sendEther(address to, uint256 amount) externa {
        (bool success,) = to.call{value: amount}("");
        require(success, "failed to send ether");
    }
}
```

- Run this test case:

```
function test_cannotReceiveEth() public {
    SomeSenderContract senderImpl = new SomeSenderContract();
    Atlas atlasImpl = new Atlas();
    uint256 fundsToSend = 0.1 ether;
    uint256 aliceInitialBalance = alice.addr.balance;

    vm.deal(bob.addr, 1 ether);
    vm.deal(address(senderImpl), fundsToSend);

    Vm.SignedDelegation memory delegation = vm.signDelegation(address(atlasImpl),
    ↪  alice.privateKey);

    vm.startBroadcast(bob.privateKey);
    vm.attachDelegation(delegation);
    assertGt(alice.addr.code.length, 0, "Alice should have delegation code");

    Atlas.Call memory call = IAtlas.Call({
        to: address(senderImpl),
        value: 0,
        data: abi.encodeCall(senderImpl.sendEther, (alice.addr, fundsToSend))
    });

    uint256 deadline = block.timestamp + 10_000;
    uint256 cnonce = vm.randomUint();

    bytes32 digest = getDigest(call, deadline, cnonce);
    (uint8 v, bytes32 r, bytes32 s) = vm.sign(alice, digest);

    Atlas(alice.addr).executeCall(call, deadline, cnonce, v, r, s); // <-- THIS
    ↪  REVERTS

    assertEq(alice.addr.balance, aliceInitialBalance + fundsToSend, "wrong
    ↪  balance");
}
```

Recommendation:

- Implement the above-mentioned functions correctly.

- Or alternatively, inherit from Receiver contract and modify as per needs.

Byzantine Finance: Fixed in PR 8.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Signature replay due to shared storage in EIP-7702 context

Severity: Medium Risk

Context: Atlas.sol#L44

Description: The `usedNonces` mapping uses standard storage slots, which can be corrupted when users switch EIP-7702 delegations, enabling replay of previously consumed signatures. In EIP-7702, storage lives on the delegating EOA, not the implementation contract. When a user changes delegation (Atlas → OtherImpl → Atlas), the intermediate implementation may overwrite mapping slots for its own purposes, resetting nonce values to false. This allows previously-executed non-expired signature to be replayed.

Recommendation: Implement EIP-7201 namespaced storage as recommended by EIP-7702.

```
/// @custom:storage-location erc7201:atlas.storage.main
struct AtlasStorage {
    mapping(uint256 => bool) usedNonces;
}

// keccak256(abi.encode(uint256(keccak256("atlas.storage.main")) - 1)) &
↪  ~bytes32(uint256(0xff))
bytes32 private constant ATLAS_STORAGE_LOCATION = 0x...;

function _getStorage() private pure returns (AtlasStorage storage $) {
    assembly { $.slot := ATLAS_STORAGE_LOCATION }
}
```

This places `usedNonces` at a collision-resistant storage location, preventing accidental overwrites by other delegation targets. However, it is possible for malicious & untrusted delegation upgrades to re-write slots adversarially.

Byzantine Finance: Fixed in PR 7.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Atlas delegation does not implement EIP-1271 `isValidSignature`

Severity: Low Risk

Context: Atlas.sol#L32

Description: Atlas is designed as an EIP-7702 delegation target, allowing EOAs to delegate to it for batch call functionality. However, the contract does not implement the EIP-1271 `isValidSignature(bytes32 hash, bytes signature)` function.

When an EOA delegates to a contract via EIP-7702, that EOA now has code. Many protocols check `address.code.length > 0` to determine if an address is a contract and, if so, call `isValidSignature` to validate signatures rather than using ecrecover. Due to this, when `isValidSignature` is called on Atlas, the batch execution will revert.

Recommendation:

- Considering adding supported implementation for EIP-1271.
- Document the absence of the same and warn users to not interact with any callee which might call this hook.

Byzantine Finance: Fixed in commit 5857c0ef.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Unnecessary zero-address check on recovered address

Severity: Gas Optimization

Context: Atlas.sol#L67, Atlas.sol#L99

Description: The check `recoveredAddress != address(0)` is unnecessary, as it's followed by `recoveredAddress == address(this)`. Furthermore, if using OpenZeppelin's ECDSA algorithm with `recover(digest, v, r, s)`, the result will not be `0`, as the `recover()` function will throw an error in this case.

Recommendation:

```
- require(recoveredAddress != address(0) && recoveredAddress == address(this),
↪  InvalidSigner());
+ require(recoveredAddress == address(this), InvalidSigner());
```

Byzantine Finance: Fixed in PR 9.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Uninformative logging and no logging of return data

Severity: Informational

Context: Atlas.sol#L129-L131

Description: After each call, a `CallExecuted` event is logged with the input data to the call. However, this data is already transparently present in the transaction data and can thus be picked up by any observer checking for logs. Meanwhile, the returndata of the call is discarded, making it difficult to retrieve. This means on the one hand a gas cost burden of emitting uninteresting data (which also tends to be quite large, as meaningful calldata on average is larger than meaningful returndata) and extra burden on observers to retrieve interesting returndata, if they care about it.

Recommendation:

```
- (bool success,) = callItem.to.call{value: callItem.value}(callItem.data);
+ (bool success, bytes memory returndata) = callItem.to.call{value:
↪  callItem.value}(callItem.data);
  require(success, CallReverted());
- emit CallExecuted(msg.sender, callItem.to, callItem.value, callItem.data);
+ emit CallExecuted(msg.sender, callItem.to, returndata);
```

Byzantine Finance: Fixed in PR 10.

Cantina Managed: Fix verified.

3.5.2 Missing fields in EIP712 domain

Severity: Informational

Context: Atlas.sol#L37

Description: The EIP712 `DOMAIN_TYPEHASH` doesn't include `name` and `version` fields, due to this, certain wallet might display it opaquely or even reject it.

Recommendation: Consider adding `name` and `version` for wallet UX improvements.

Byzantine Finance: Fixed in PR 5.

Cantina Managed: Fix verified.