Aqua Protocol Audit Report

Version 1.0

Vadim Fadeev

December 26, 2025

Aqua Protocol Audit Report

Vadim Fadeev

December 26, 2024

Aqua Protocol Audit Report

Prepared by: Vadim Fadeev Lead Auditors:

- Vadim Fadeev

Assisting Auditors:

- None

Table of contents

See table

  - Issues found
- Findings
  - Medium
    * [M-1] `Aqua::ship` can be called multiple times with different token sets for the same strategy
  - Low
    * [L-1] ReentrancyGuard pragma version mismatch with transient storage requirements
    * [L-2] `Aqua::dock` behavior undefined when empty tokens array is provided
  - Informational
    * [I-1] Shipped event documentation incorrectly states "revoked" instead of "shipped"
    * [I-2] Inconsistent terminology: "App" vs "Strategy Address" throughout codebase
    * [I-3] Pushed event mixes "strategy" and "app" terminology
    * [I-4] `safeBalances()` does not document specific revert reasons
    * [I-5] Custom errors should be defined in the interface
    * [I-6] Typo in documentation: "abi enocoded" should be "abi encoded"
    * [I-7] Missing NatSpec documentation for Balance library
    * [I-8] Missing NatSpec documentation for Multicall contract
    * [I-9] Consider using OpenZeppelin's ReentrancyGuardTransient

About Vadim Fadeev

Disclaimer

The Vadim Fadeev team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

Risk Classification

Likelihood	High Impact	Medium Impact	Low Impact
High	H	H/M	M

Likelihood	High Impact	Medium Impact	Low Impact
Medium	H/M	M	M/L
Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

```
[0c908761cf9f9b18ada86a6ef8b72c8534dc8005]
```

Scope

```
src/├──
 Aqua.sol├──
 AquaApp.sol├──
 AquaRouter.sol├──
 interfaces/│ └──
    IAqua.sol└──
 libs/├──
     Balance.sol├──
     Multicall.sol├──
     ReentrancyGuard.sol├──
     Simulator.sol├──
     Transient.sol└──
     TransientLock.sol
```

Protocol Summary

Aqua is a shared liquidity layer protocol that manages token balances (allowances) between makers (liquidity providers) and apps. The protocol enables shared liquidity access directly from maker wallets, allowing apps/strategies to pull tokens from makers while maintaining balance tracking.

Roles

- **Maker**: Liquidity provider who ships strategies and grants token allowances to apps

- **App**: Strategy implementation contract that can pull tokens from makers based on granted balances

Executive Summary

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	2
Info	9
Gas Optimizations	0
Total	12

Findings

Medium

[M-1] `Aqua::ship` can be called multiple times with different token sets for the same strategy

Description: The `Aqua::ship` function allows a maker to ship a strategy with a set of tokens. The function checks if each token's `tokensCount` is 0 to prevent re-shipping the same strategy. However, the check is performed per-token, not per-strategy.

```
function ship(address app, bytes calldata strategy, address[] calldata
    tokens, uint256[] calldata amounts) external returns(bytes32
    strategyHash) {
     strategyHash = keccak256(strategy);
     uint8 tokensCount = tokens.length.toUint8();
     require(tokensCount != _DOCKED, MaxNumberOfTokensExceeded(
         tokensCount, _DOCKED));

     emit Shipped(msg.sender, app, strategyHash, strategy);
     for (uint256 i = 0; i < tokens.length; i++) {
         Balance storage balance = _balances[msg.sender][app][
             strategyHash][tokens[i]];
@>      require(balance.tokensCount == 0, StrategiesMustBeImmutable(app
    , strategyHash));
         balance.store(amounts[i].toUint248(), tokensCount);
         emit Pushed(msg.sender, app, strategyHash, tokens[i], amounts[i
             ]);
     }
}
```

This means a maker can call `ship` multiple times for the same `strategyHash` with different token arrays. For example: 1. First call: `ship(app, strategy, [tokenA, tokenB], [100, 200])` 2. Second call: `ship(app, strategy, [tokenC, tokenD], [300, 400])`

Both calls would succeed because `tokenC` and `tokenD` have `tokensCount == 0`.

Impact: This could lead to inconsistent state where a single strategy has multiple token sets with different `tokensCount` values, potentially causing issues with `dock()` validation and overall strategy management.

Proof of Concept:

Add the following to the `Aqua.t.sol` test suite:

```
function test_ship_can_be_called_twice_different_tokens() public {
    address app = address(0x1);
    bytes memory strategy = "test_strategy";

    address[] memory tokens1 = new address[](2);
    tokens1[0] = address(tokenA);
    tokens1[1] = address(tokenB);

    uint256[] memory amounts1 = new uint256[](2);
    amounts1[0] = 100;
    amounts1[1] = 200;

    // First ship succeeds
    aqua.ship(app, strategy, tokens1, amounts1);

    address[] memory tokens2 = new address[](2);
    tokens2[0] = address(tokenC);
    tokens2[1] = address(tokenD);

    uint256[] memory amounts2 = new uint256[](2);
    amounts2[0] = 300;
    amounts2[1] = 400;

    // Second ship with different tokens also succeeds
    aqua.ship(app, strategy, tokens2, amounts2);
}
```

Recommended Mitigation: Track shipped strategies at the strategy level, not per-token. Consider adding a separate mapping to track whether a strategy has been shipped:

```
mapping(address maker => mapping(address app => mapping(bytes32
    strategyHash => bool))) private _shippedStrategies;

function ship(address app, bytes calldata strategy, address[] calldata
    tokens, uint256[] calldata amounts) external returns(bytes32
    strategyHash) {
    strategyHash = keccak256(strategy);
    require(!_shippedStrategies[msg.sender][app][strategyHash],
        StrategiesMustBeImmutable(app, strategyHash));
    _shippedStrategies[msg.sender][app][strategyHash] = true;
    // ... rest of the function
}
```

Low

[L-1] ReentrancyGuard pragma version mismatch with transient storage requirements

Description: The `ReentrancyGuard.sol` contract specifies `pragma solidity ^0.8.0` but uses transient storage operations (`tload`/`tstore`) which are only available since Solidity 0.8.24.

```
// SPDX-License-Identifier: LicenseRef-Degensoft-Aqua-Source-1.1
pragma solidity ^0.8.0; // tload/tstore are available since 0.8.24
```

While there is a comment noting this, the pragma itself allows compilation with incompatible versions.

Impact: If compiled with Solidity versions < 0.8.24, the contract will fail at runtime when attempting to use transient storage operations.

Recommended Mitigation: Update the pragma to enforce the minimum required version:

```
- pragma solidity ^0.8.0; // tload/tstore are available since 0.8.24
+ pragma solidity ^0.8.24;
```

Also add comprehensive NatSpec documentation:

```
/// @title ReentrancyGuard - Transient Storage-Based Protection
/// @notice Abstract contract providing reentrancy guards using
    transient storage for gas efficiency.
/// @dev Requires Solidity >=0.8.24 for transient storage support.
abstract contract ReentrancyGuard {
    // ...
}
```

[L-2] Aqua::dock behavior undefined when empty tokens array is provided

Description: The `dock()` function documentation states "Sets balances to 0 for all specified tokens," but does not specify what happens when an empty `tokens` array is provided.

```
function dock(address app, bytes32 strategyHash, address[] calldata
    tokens) external {
    for (uint256 i = 0; i < tokens.length; i++) {
        Balance storage balance = _balances[msg.sender][app][
            strategyHash][tokens[i]];
        require(balance.tokensCount == tokens.length,
            DockingShouldCloseAllTokens(app, strategyHash));
        balance.store(0, _DOCKED);
    }
    emit Docked(msg.sender, app, strategyHash);
}
```

If `tokens.length == 0`, the loop is skipped, the `Docked` event is emitted, but no actual docking occurs. This could lead to confusion as the event suggests the strategy was docked when it actually wasn't.

Impact: Potential confusion for off-chain systems monitoring `Docked` events. A strategy could appear docked without actually being deactivated.

Recommended Mitigation: Add a check to prevent docking with an empty tokens array:

```
error EmptyTokensArray();

function dock(address app, bytes32 strategyHash, address[] calldata
    tokens) external {
    require(tokens.length > 0, EmptyTokensArray());
    // ... rest of function
}
```

Informational

[I-1] Shipped event documentation incorrectly states "revoked" instead of "shipped"

Description: In the `IAqua.sol` interface, the `Shipped` event documentation for the `app` parameter incorrectly states "The strategy address being revoked":

```
/// @notice Emitted when a new strategy is shipped (deployed) and
    initialized with balances
/// @param maker The address of the maker shipping the strategy
@> /// @param app The strategy address being revoked
/// @param strategyHash The hash of the strategy being shipped
/// @param strategy The strategy being shipped (abi enocoded)
event Shipped(address maker, address app, bytes32 strategyHash, bytes
    strategy);
```

This appears to be a copy-paste error from the `Docked` event where "revoked" is appropriate.

Recommended Mitigation: Update the documentation:

```
- /// @param app The strategy address being revoked
+ /// @param app The app/strategy address being shipped
```

[I-2] Inconsistent terminology: "App" vs "Strategy Address" throughout codebase

Description: Throughout the codebase, the `app` parameter is referred to inconsistently as: - "the strategy address" (in events) - "the app/strategy" (in function comments) - "the implementation contract" (in `ship()`)

Recommended Mitigation: Standardize terminology. Define "app" clearly at the interface level and use it consistently:

```
/// @notice Manages token balances between makers and apps
/// @dev An "app" is an implementation contract that contains strategy
    logic and can pull tokens from makers
interface IAqua {
```

[I-3] Pushed event mixes "strategy" and "app" terminology

Description: In the `Pushed` event documentation, the `app` parameter is described as "The strategy that gets increased balance":

```
/// @param app The strategy that gets increased balance
```

This mixes terminology between "app" (the parameter name) and "strategy" (the description).

Recommended Mitigation: Use consistent terminology:

```
- /// @param app The strategy that gets increased balance
+ /// @param app The app whose balance is increased
```

[I-4] `safeBalances()` does not document specific revert reasons

Description: The `safeBalances()` function documentation mentions it “reverts if any of the tokens is not part of the active strategy” but does not specify the custom error that will be thrown.

Recommended Mitigation: Add the error to the documentation:

```
/// @notice Returns balances of multiple tokens in a strategy
/// @dev Reverts with SafeBalancesForTokenNotInActiveStrategy if token
    is not in active strategy
```

[I-5] Custom errors should be defined in the interface

Description: The `IAqua` interface does not define the custom errors used by the implementation. The errors are defined in `Aqua.sol` but not exposed in the interface, making it harder for integrators to programmatically handle errors.

The following errors are used in the implementation but missing from the interface:

```
error MaxNumberOfTokensExceeded(uint256 tokensCount, uint256
    maxTokensCount);
error StrategiesMustBeImmutable(address app, bytes32 strategyHash);
error DockingShouldCloseAllTokens(address app, bytes32 strategyHash);
error PushToNonActiveStrategyPrevented(address maker, address app,
    bytes32 strategyHash, address token);
error SafeBalancesForTokenNotInActiveStrategy(address maker, address
    app, bytes32 strategyHash, address token);
```

Recommended Mitigation: Move the custom error definitions to the `IAqua` interface and add NatSpec documentation explaining when each error is thrown:

```
interface IAqua {
    /// @notice Thrown when attempting to ship a strategy with more
        than 254 tokens
    error MaxNumberOfTokensExceeded(uint256 tokensCount, uint256
        maxTokensCount);

    /// @notice Thrown when attempting to ship a strategy that has already been shipped
    error StrategiesMustBeImmutable(address app, bytes32 strategyHash);

    /// @notice Thrown when docking with incorrect number of tokens
    error DockingShouldCloseAllTokens(address app, bytes32 strategyHash);

    /// @notice Thrown when pushing to a non-active or docked strategy
    error PushToNonActiveStrategyPrevented(address maker, address app, bytes32 strategyHash, address token);

    /// @notice Thrown when querying safe balances for a token not in the active strategy
    error SafeBalancesForTokenNotInActiveStrategy(address maker, address app, bytes32 strategyHash, address token);

    // ... rest of interface
}
```

[I-6] Typo in documentation: “abi enocoded” should be “abi encoded”

Description: In the `Shipped` event documentation:

```
/// @param strategy The strategy being shipped (abi enocoded)
```

Recommended Mitigation: Fix the typo:

```diff
- /// @param strategy The strategy being shipped (abi enocoded)
+ /// @param strategy The strategy being shipped (abi encoded)
```

[I-7] Missing NatSpec documentation for Balance library

Description: The `Balance.sol` library lacks library-level NatSpec documentation explaining its purpose and the struct packing scheme.

Recommended Mitigation: Add comprehensive documentation:

```
/// @title Balance Library
/// @notice Library for efficient storage of Balance structs by packing into a single slot
/// @dev Uses 248 bits for amount (sufficient for token balances) and 8
     bits for tokensCount
struct Balance {
    uint248 amount;
    uint8 tokensCount;
}
```

[I-8] Missing NatSpec documentation for Multicall contract

Description: The `Multicall.sol` contract lacks NatSpec documentation for the contract and its function.

Recommended Mitigation: Add documentation:

```
/// @title Multicall - Batch Execution Utility
/// @notice Allows batching multiple calls to this contract via
    delegatecall for atomic operations
/// @dev Intended for inheritance. Be cautious of recursion in batched
    calldata.
contract Multicall {
    /// @notice Execute multiple calls in a single transaction
    /// @param data Array of encoded function calls
    function multicall(bytes[] calldata data) external {
        // ...
    }
}
```

[I-9] Consider using OpenZeppelin's ReentrancyGuardTransient

Description: The protocol implements a custom `ReentrancyGuard` using transient storage. OpenZeppelin provides a battle-tested implementation (`ReentrancyGuardTransient`) that could be used instead.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/308c4914b5482e8ffcbbe6f00be7d7f04298
tracts/utils/ReentrancyGuardTransient.sol

Impact: Using a custom implementation increases audit surface and maintenance burden.

Recommended Mitigation: Evaluate whether OpenZeppelin's `ReentrancyGuardTransient` meets the protocol's requirements. If the custom implementation provides specific features not available in OpenZeppelin's version, document these differences clearly.