Avon Protocol Audit Report

Auction Liquidations

Prepared by: alix40, Senior Security Researcher

Date: 30.09.2025

Commit:
b06e021d6d1891c22db50f07ab11da2cdf596440

About

Valkyri Security is an elite **full-stack** security firm specializing in Web3 auditing, blockchain vulnerability analysis, smart contract audits, as well as web & cloud penetration testing. They have secured billions in protocol assets, completed hundreds of audits, and found critical vulnerabilities across both Web2 and Web3 ecosystems. (valkyrisec.com)

About Avon

Avon is a decentralized lending protocol that implements an orderbook-based system for matching lenders and borrowers. The core components include OrderbookFactory for creating and managing orderbooks, Orderbook for managing lending/borrowing orders using red-black trees, and Pool contracts implementing ERC-4626 for managing deposits, borrowing, and liquidations.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - leads to a moderate material loss of assets in the protocol or moderately harms a group of users.
- Low - leads to a minor material loss of assets in the protocol or harms a small group of users.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Action required for severity levels

- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

Executive Summary

This section gives a high-level overview of all the identified bugs/weaknesses (QA-01 to QA-04) and recommended remediations.

Bug ID	Title / Short Description	Status
QA-01	No external query for current `transientLiquidationBonus`	Fixed
QA-02	Interface mismatch: functions missing in `IPoolImplementation.sol`	Fixed
QA-03	Griefing possible during auction via repay of dust amounts	Acknowledged
QA-04	Insufficient oracle-selector whitelist validation	Fixed

These are the major findings. The rest of the report confirms correct implementations around new variables, roles, methods, and mitigation of additional attack vectors.

Verifications & Checks

New `auctionPriorityWindow` State Variable

We verified:
- It is properly added and configured via factory and pool constructor.
- Validation ensures it does not exceed `PoolConstants.MAX_AUCTION_PRIORITY_WINDOW`.
- In liquidation, non-auction liquidations are blocked during the priority window.
- A window of `0` disables blocking.
- It is queryable via `getAuctionPriorityWindow()`.

New Auction Role

We verified:
- Existence of `AUCTION_ROLE` in contract code.
- Role is managed by `ProposerRole`.
- `grantRole` / `revokeRole` applicable.
- Supplementary management via `grantAuctionRole()` / `revokeAuctionRole()` by pool manager & proposer.
- Only accounts holding `AUCTION_ROLE` can call `updateTransientLiquidationBonus()`.

New Method: `updateTransientLiquidationBonus`

We verified:
- Use of transient storage (`tload() / tstore()`).
- Addition to interface definitions.
- Restricted access via `onlyRole(AUCTION_ROLE)`.
- Fallback logic to persistent liquidation bonus if transient is unset.
- Correct validation logic.
- Proper effect in liquidation flows.

New Method: `configureAuctionPriorityWindow()`

We verified:
- Only `ProposerRole` or `PoolManager` can call it.
- Configuration logic correctly updates the window.
- Non-auction liquidations are blocked during the window.

New Control Contract: `AvonLiquidationBonusProposer`

We verified:
- Access control is strictly enforced on external/public endpoints.
- Whitelisting of selectors is implemented.
- Solver logic correctness.
- Only whitelisted oracles are accepted for updates.

Mitigated Attack Vectors

- Cross-pool liquidation during price changes: mitigated by per-pool oracle feeds and restricted updates (Assumptions 01/02).
- Cross-pool liquidation during interest accrual: mitigated by blocking non-auction liquidations during the priority window unless transient bonus is set.

Deployment Assumptions

- **Assumption-01:** Each pool must have its own distinct price feed.
- **Assumption-02:** `AvonLiquidationBonusProposer` may only update price for exactly one pool.

Findings & Recommendations (Detailed)

QA-01: No External Query for `transientLiquidationBonus`

Description:
External contracts or clients can't read the current `transientLiquidationBonus`, reducing visibility and auditability during auctions.

Recommendation:
Add a public / external getter, e.g.

```
function getTransientLiquidationBonus() external view returns (uint256) {
    return PoolStorage.getTransientLiquidationBonus();
}
```

QA-02: Missing Pool Interface Methods

Description:
Some new functions in `AvonPool.sol` are absent in the interface `IPoolImplementation.sol`, causing a mismatch.

Recommendation:
Extend the interface by adding:

```
function grantAuctionRole(address account) external;
function revokeAuctionRole(address account) external;
```

QA-03: Griefing Via Repay During Auction

Description:
During the priority window, an attacker could artificially set the liquidation bonus low, then repay small (“dust”) amounts to push positions back to healthy state, blocking others.

Recommendation:
During priority window (when `transientLiquidationBonus` is active), disable `repay()` or disallow it in that window to prevent such griefing.

QA-04: Insufficient Whitelisting Logic for Oracle + Selector

Description:
Currently, whitelist logic only considers selectors globally. If oracle B supports selector A, then oracle B could be used to call selector A even if not intended.

Recommendation:

Adopt a mapping of the form:

```
mapping(address oracle => mapping(bytes4 selector => bool isAllowed)) public allowedSelectors;
```

Then verify both oracle and selector together.
