Reservoir oracle

Security Review

Cantina Managed review by:
Desmond Ho, Lead Security Researcher
Throttle, Security Researcher

December 11, 2024

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

Reservoir Finance is dedicated to developing groundbreaking solutions that optimize capital efficiency.

From Dec 3rd to Dec 7th the Cantina team conducted a review of reservoir-oracle on commit hash ec72d854. The team identified a total of **10** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 2
- Gas Optimizations: 2
- Informational: 6

3 Findings

3.1 Low Risk

3.1.1 `route()` doesn't return a route for reverse token order

Severity: Low Risk

Context: ReservoirPriceOracle.sol#L106-L108

Description: `_getRouteDecimalDifferencePrice()` requires `aToken0 < aToken1` because this determines the storage slot lookup for the route. Hence, in the case where the reverse order is queried, where `aToken0 > aToken1`, there will be no route returned. However, it should return the reverse route if it's been set.

Proof of Concept:

```
function testGetRoute() external {
    // setup
    address lToken0 = address(_tokenB);
    address lToken1 = address(_tokenC);
    address[] memory lRoute = new address[](2);
    uint16[] memory lRewardThreshold = new uint16[](1);
    lRewardThreshold[0] = Constants.BP_SCALE;
    lRoute[0] = lToken0;
    lRoute[1] = lToken1;
    _oracle.setRoute(lToken0, lToken1, lRoute, lRewardThreshold);

    // query route with ordered tokens, assert non-zero route
    address[] memory rRoute = _oracle.route(lToken0, lToken1);
    assertNotEq(rRoute.length, 0);

    // query reverse token order, no route returned
    rRoute = _oracle.route(lToken1, lToken0);
    assertEq(rRoute.length, 0);
}
```


Recommendation:

```
function route(address aToken0, address aToken1) external view returns (address[] memory rRoute) {
   (address lToken0, address lToken1) = Utils.sortTokens(aToken0, aToken1);
   (rRoute,,,) = _getRouteDecimalDifferencePrice(lToken0, lToken1);
   if (lToken0 != aToken0) rRoute.reverse();
}
```

Reservoir-Labs: Fixed in commit 23d563b4.

Cantina Managed: Fixed. Implementation remains unchanged, but natspec was added requiring the token order to be sorted.

3.1.2 Unexpected effects when directly overriding a route instead of clearing and re-setting

Severity: Low Risk

Context: ReservoirPriceOracle.sol#L486

Description: There are a couple of undesired effects should an existing route be directly overridden instead of clearing and re-setting it.

- 2nd slot remains dirty when shortening max hop routes: If a max hop route is directly overwritten to a shorter route without clearing it, the 2nd slot will remain dirty. This however shouldn't pose a problem because the route type has been updated to a shorter one, so the dirty slot shouldn't be read until updated to a max hop route again.

  Proof of Concept:

```
function testSecondSlotRemainsDirty() external {
    // set intermediate route with max hop
    address lStart = address(_tokenA);
    address lIntermediate1 = address(_tokenC);
    address lIntermediate2 = address(_tokenB);
    address lEnd = address(_tokenD);
    address[] memory lRoute = new address[](4);
    lRoute[0] = lStart;
    lRoute[1] = lIntermediate1;
    lRoute[2] = lIntermediate2;
    lRoute[3] = lEnd;
    uint16[] memory lRewardThreshold = new uint16[](3);
    lRewardThreshold[0] = lRewardThreshold[1] = lRewardThreshold[2] = Constants.BP_SCALE;
    _oracle.setRoute(lStart, lEnd, lRoute, lRewardThreshold);

    // then overwrite to a shorter route
    lRoute = new address[](3);
    lRoute[0] = lStart;
    lRoute[1] = lIntermediate1;
    lRoute[2] = lEnd;
    lRewardThreshold = new uint16[](2);
    lRewardThreshold[0] = lRewardThreshold[1] = Constants.BP_SCALE;
    _oracle.setRoute(lStart, lEnd, lRoute, lRewardThreshold);

    // check that the 2nd slot remains dirty
    bytes32 lSecondSlot = bytes32(uint256(keccak256(abi.encode(lStart, lEnd))) + 1);
    bytes32 secondSlotValue = vm.load(address(_oracle), lSecondSlot);
    address thirdToken = address(bytes20(secondSlotValue));
    assertEq(thirdToken, lIntermediate2);
}
```

- Silent update skip: Some slot might silently end up with obsolete data. For example, initial setting where pair A/C with route A → B → C ends up with following bytes:

```
slot(A,C) := [0x02, bytes20(address(B)), 11 * 0x00]
slot(A,B) := [0x01, diff, RT, price_A_B]
slot(B,C) := [0x01, diff, RT, price_B_C]
```

Then after `setRoute(A, D, [A,C,D], ..)` is called, the update of `slot(A,C)` will be skipped in function `_checkAndPopulateIntermediateRoute`. The storage layout will end up like this:

```
slot(A,D) := [0x02, bytes20(address(C)), 11 * 0x00]
slot(A,C) := [0x02, bytes20(address(B)), 11 * 0x00]
slot(C,D) := [0x01, diff, RT, price_C_D]
```

But should be something like:

```
slot(A,D) := [0x02, bytes20(address(C)), 11 * 0x00]
slot(A,C) := [0x01, diff, RT, price_A_C]
slot(C,D) := [0x01, diff, RT, price_C_D]
```

Proof of Concept:

```
function testDirtySlot() external {
    // set pair A/C with route [A->B->C]
    address lStart = address(_tokenA);
    address lIntermediate1 = address(_tokenB);
    address lEnd = address(_tokenC);
    address[] memory lRoute = new address[](3);
    lRoute[0] = lStart;
    lRoute[1] = lIntermediate1;
    lRoute[2] = lEnd;
    uint16[] memory lRewardThreshold = new uint16[](2);
    lRewardThreshold[0] = lRewardThreshold[1] = Constants.BP_SCALE;
    _oracle.setRoute(lStart, lEnd, lRoute, lRewardThreshold);

    // Test if A/C slot contains 2-HOP route as expected
    bytes32 AC_Slot = bytes32(uint256(keccak256(abi.encode(address(_tokenA), address(_tokenC)))));
    bytes32 AC_SlotValue = vm.load(address(_oracle), AC_Slot);
    assertEq(AC_SlotValue.is2HopRoute(), true);

    // Overwrite with pair A/D with route [A->C->D]
    lEnd = address(_tokenD);
    lRoute = new address[](3);
    lRoute[0] = lStart;
    lRoute[1] = address(_tokenC);
    lRoute[2] = lEnd = address(_tokenD);
    _oracle.setRoute(lStart, lEnd, lRoute, lRewardThreshold);

    // Test if pair A/C is now of type simple, not 2-HOP
    AC_SlotValue = vm.load(address(_oracle), AC_Slot);
    bool IS_FIXED = false;
    if (IS_FIXED) {
        assertEq(AC_SlotValue.isSimplePrice(), true, "slot(A,C) is not a of type SimplePrice, but
↪  should be");
    } else {
        assertEq(AC_SlotValue.isSimplePrice(), false, "slot(A,C) is of type SimplePrice, but shouldn't
↪  be");
    }
}
```

Recommendation: It is advisable to clear the route before setting a new one, instead of directly overwriting.

Reservoir-Labs: Fixed in commit 272b674d and 6ce37dd1.

Cantina Managed: Fixed. `clearRoute()` is automatically called if there is an existing route, before the new route is set.

```
bytes32 lSlotData;
assembly ("memory-safe") {
    lSlotData := sload(lSlot)
}
// clear existing route first if there is one
if (lSlotData != 0) clearRoute(aToken0, aToken1);
```

Also, `_checkAndPopulateIntermediateRoute()` checks that the intermediate routes to be used are simple routes.

3.2 Gas Optimization

3.2.1 Storage variables can be more tightly packed

Severity: Gas Optimization

Context: ReservoirPriceOracle.sol#L51-L64

Description: Currently, `rewardGasAmount` and `fallbackOracle` are packed together in a single storage slot, as shown in the output of `forge inspect ReservoirPriceOracle storage --pretty`. This leaves room for potential optimization.

Name	Type	Slot	Offset	Bytes
`owner`	`address`	0	0	20
`fallbackOracle`	`address`	1	0	20
`rewardGasAmount`	`uint64`	1	20	8
`twapPeriod`	`uint64`	2	0	8
`pairs`	`mapping(address => mapping(address => contract ReservoirPair))`	3	0	32

Recommendation: Since `MAX_TWAP_PERIOD` is `3600 < type(uint16).max`, `twapPeriod` can be defined as `uint16` instead of `uint64`, allowing all 3 variables (`fallbackOracle`, `rewardGasAmount` and `twapPeriod`) to be packed in a single slot.

Reservoir-Labs: Fixed in commit a098bb92.

Cantina Managed: Fixed.

3.2.2 Reward threshold can be denominated in WAD instead of basis points

Severity: Gas Optimization

Context: ReservoirPriceOracle.sol#L508-L510

Description: The reward threshold is denominated in basis points, but is scaled to `WAD` for reward calculations. This scaling becomes redundant if the reward threshold is denominated in `WAD` instead.

Recommendation: There is sufficient space to store the reward threshold in `WAD`, by allocating 8 bytes to it. The new packing arrangement will be

- 1 byte for the flag type: `FLAG_SIMPLE_PRICE`.
- 1 byte for decimal difference.
- 8 bytes = `type(uint64).max` > `1e18` requirement for reward threshold.
- 22 bytes = `type(uint176).max` > `type(uint128).max` requirement for price.

Reservoir-Labs: Fixed in commit 1cbb74c7 and 272b674d (update natspec).

Cantina Managed: Fixed.

3.3 Informational

3.3.1 Missing natspec

Severity: Informational

Context: ReservoirPriceOracle.sol#L106

Description: `route()` is the only external function lacking natspec. Having it would improve code readability and assist integrators.

Recommendation: Add the natspec for the `route()` function.

Reservoir-Labs: Fixed in commit 23d563b4.

Cantina Managed: Fixed.

3.3.2 Redundant assert

Severity: Informational

Context: ReservoirPriceOracle.sol#L225-L227

Description: The previous conditionals `if (lPercentDiff < lRewardThresholdWAD)` and `else if (lPercentDiff >= lRewardThresholdWAD * MAX_REWARD_MULTIPLIER)` means that `lPercentDiff` has to between `lRewardThresholdWAD` (inclusive) and `lRewardThresholdWAD * MAX_REWARD_MULTIPLIER` (exclusive), which is the assertion made.

It is therefore a redundant assertion.

Recommendation: Remove the assertion. To an extent, the other 2 assertions in the contract are also redundant, but would be good to be left in for safety.

Reservoir-Labs: Fixed in ea6fa621.

Cantina Managed: Fixed.

3.3.3 Uncaught revert from decoding return data from ill-formed tokens

Severity: Informational

Context: ReservoirPriceOracle.sol#L364-L369

Description: The `try-catch` has another case where a revert won't be caught in the `catch` clause, which is when the abi-decoding of the return data fails.

Proof of Concept:

```
contract BadERC4626 {
  function asset() external pure returns (bool) {
      assembly {
          mstore(0x0, 0x1) // Store `true` in memory
          return(0x0, 0x1) // Return only 1 byte instead of the full 32 bytes expected of return data
      }
  }
}

function testRevertFromBadERC4626() external {
   BadERC4626 badToken = new BadERC4626();
   _oracle.getQuote(1000, address(badToken), address(_tokenA));
}
```

Recommendation: Consider modifying the comment to add this case in.

Reservoir-Labs: Fixed in commit 97d95ff9.

Cantina Managed: Fixed.

3.3.4 Redundancies

Severity: Informational

Context: Constants.sol#L9, OracleErrors.sol#L15

Description: The referenced lines are defined variables but are unused.

Recommendation: Remove the redundancies.

Reservoir-Labs: Fixed in commit ddaca12c.

Cantina Managed: Fixed.

3.3.5 Comment Improvement

Severity: Informational

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

Description: The maximum price supported is `Constants.MAX_SUPPORTED_PRICE = type(uint128).max`.

Recommendation:

```
- // Assumes that aPrice is between 1 and 1e36
+ // Assumes that aPrice is between 1 and Constants.MAX_SUPPORTED_PRICE
```

Reservoir-Labs: Fixed in commit adbe5511.

Cantina Managed: Fixed.

3.3.6 Nested imports

Severity: Informational

Context: ReservoirPriceOracle.sol#L10

Description: Certain imports are nested, which could reduce code readability and maintainability. For instance:

- `PriceType` could be imported directly from `src/Enum.sol`.
- `ReservoirPair` could be imported directly from `lib/amm-core/src/ReservoirPair.sol`.

Recommendation: Consider refactoring these imports to reference their original sources directly.

Reservoir-Labs: Acknowledged.

Cantina Managed: Acknowledged.