Risk Oracle Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Giovanni Di Siena

September 9, 2024

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse 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.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Chaos Labs' platform revolutionizes on-chain risk management, security, and incentive strategies by automating real-time protocol parameter recommendations, utilizing sophisticated data analytics, modeling, and simulations. Chaos Labs operate an off-chain Risk Engine – when it identifies a new recommendation, this will be written directly to the Risk Oracle smart contract for consumption by integrating protocols. Chaos Labs will be responsible for regularly monitoring the state of the integrating protocols and automating parameter recommendations.

5 Audit Scope

Cyfrin conducted an audit of the Chaos Labs Risk Oracle smart contract based on the code present in the repository commit hash 9449219.

The `RiskOracle.sol` contract and its dependencies were included in the scope of the audit.

6 Executive Summary

Over the course of 6 days, the Cyfrin team conducted an audit on the Risk Oracle smart contracts provided by Chaos Labs. In this period, a total of 8 issues were found.

This review of the Chaos Labs Risk Oracle contracts yielded one medium-severity scenario in which incorrect state updates could occur, likely resulting in integrating contracts consuming inaccurate historical data.

The test suite was comprehensive, covering all core functionalities of the contract, and it has since been bolstered by the addition of more complex unit and fuzz tests that are sufficient to catch the issues identified in this review. A stateful fuzz testing suite has also been added, which can be extended in the future as the complexity of this and any other contract(s) increases.

Additionally, due to the use of more recent Solidity versions as specified by the `^0.8.25` pragma directive, the technique documented in this StackExchange Ethereum post was used to verify that the `PUSH0` opcode is supported

by all EVM chains on which the contract is intended to be deployed. This should be extended to checking other newly-supported opcodes, such as `TSTORE/TLOAD`, if the contract or any of its dependencies are modified in such a way that necessitates support for them.

Summary

Project Name	Risk Oracle
Repository	risk-oracle
Commit	9449219174e3...
Audit Timeline	Aug 14th - Aug 21st
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	0
Informational	5
Gas Optimizations	2
Total Issues	8

Summary of Findings

[M-1] Incorrect state update in `RiskOracle::_processUpdate`	Resolved
[I-1] Asymmetry in validation between `RiskOracle::addUpdateType` and contract constructor	Resolved
[I-2] No restriction on the contract owner becoming an authorized sender	Acknowledged
[I-3] Duplicated validation can be moved to shared internal function	Resolved
[I-4] Parallel data structures are not necessary	Resolved
[I-5] Unreachable code can be removed	Resolved
[G-1] Unnecessary initialization can be removed	Resolved
[G-2] Array length validation is not necessary	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Incorrect state update in `RiskOracle::_processUpdate`

Description: Within the `RiskParameterUpdate` struct, there is a field `bytes previousValue` that is intended to store the previous value of a parameter. This state update is performed within `RiskOracle::_processUpdate` with the value obtained from the `updatedById` mapping; however, there is no differentiation between parameters for different update types and markets so this state update will be inaccurate with overwhelming likelihood when there are multiple update types/markets. As such, consumers of this contract will receive recommendations with previous values that could be wildly different from what is expected and perhaps execute risk parameter updates based on a delta that is not representative of the real change.

Impact: The `previousValue` state for a given update will be incorrect with a very high likelihood and could result in consumers making risk parameter updates based on inaccurate historical data.

Proof of Concept: The following test was written to demonstrate this finding and has since been added to the repository during this engagement.

```
function test_PreviousValueIsCorrectForSpecificMarketAndType() public {
    bytes memory market1 = abi.encodePacked("market1");
    bytes memory market2 = abi.encodePacked("market2");
    bytes memory newValue1 = abi.encodePacked("value1");
    bytes memory newValue2 = abi.encodePacked("value2");
    bytes memory newValue3 = abi.encodePacked("value3");
    bytes memory newValue4 = abi.encodePacked("value4");
    string memory updateType = initialUpdateTypes[0];

    vm.startPrank(AUTHORIZED_SENDER);

    // Publish first update for market1 and type1
    riskOracle.publishRiskParameterUpdate(
        "ref1", newValue1, updateType, market1, abi.encodePacked("additionalData1")
    );

    // Publish second update for market1 and type1
    riskOracle.publishRiskParameterUpdate(
        "ref2", newValue2, updateType, market1, abi.encodePacked("additionalData2")
    );

    // Publish first update for market2 and type1
    riskOracle.publishRiskParameterUpdate(
        "ref3", newValue3, updateType, market2, abi.encodePacked("additionalData3")
    );

    // Publish first update for market1 and type1
    riskOracle.publishRiskParameterUpdate(
        "ref4", newValue4, updateType, market1, abi.encodePacked("additionalData4")
    );

    vm.stopPrank();

    // Fetch the latest update for market1 and type1
    RiskOracle.RiskParameterUpdate memory latestUpdateMarket1Type1 =
        riskOracle.getLatestUpdateByParameterAndMarket(updateType, market1);
    assertEq(latestUpdateMarket1Type1.previousValue, newValue2);

    // Fetch the latest update for market2 and type1
    RiskOracle.RiskParameterUpdate memory latestUpdateMarket2Type1 =
        riskOracle.getLatestUpdateByParameterAndMarket(updateType, market2);
    assertEq(latestUpdateMarket2Type1.previousValue, bytes(""));
```

```
}
```

Recommended Mitigation: Retrieve the correct historical value using the `latestUpdateIdByMarketAndType` mapping.

Chaos Labs: Fixed in commit d16a227.

Cyfrin: Verified, the previous update value is now retrieved from the correct identifier.

7.2 Informational

7.2.1 Asymmetry in validation between `RiskOracle::addUpdateType` and contract constructor

Description: The following validation is present within `RiskOracle::addUpdateType`:

```
require(!validUpdateTypes[newUpdateType], "Update type already exists.");
```

However, this function has the `onlyOwner` modifier applied, so the validation is not strictly necessary. This can be observed within the constructor, invoked when the owner deploys the contract, where there is no such validation – here, it is assumed that duplicates will be checked off-chain. As such, there is an asymmetry between these two instances that is recommended to be made consistent by either completely removing the validation or having it present in both code paths.

Chaos Labs: Added duplicate check in constructor in commit 9f7375a.

Cyfrin: Verified, the duplicate check has been added to the constructor.

7.2.2 No restriction on the contract owner becoming an authorized sender

Description: Due to the presence of the `onlyOwner` modifier applied to `RiskOracle::addAuthorizedSender`, only the contract owner is permitted to add authorized senders. Currently, the only validation present is to prevent adding an authorized sender that is already authorized, so it is possible for the owner to add themselves as an authorized sender. If this is not desired, for example to strictly enforce a separation of concerns between the two roles, then this restriction should be added.

Chaos Labs: Acknowledged there is no restriction on the contract owner becoming an authorized sender.

Cyfrin: Acknowledged.

7.2.3 Duplicated validation can be moved to shared internal function

Description: Currently, both `RiskOracle::publishRiskParameterUpdate` and `RiskOracle::publishBulkRiskParameterUpdates` contain essentially the same validation:

```
// `RiskOracle::publishRiskParameterUpdate`:
require(validUpdateTypes[updateType], "Unauthorized update type.");

// `RiskOracle::publishBulkRiskParameterUpdates`:
require(validUpdateTypes[updateTypes[i]], "Unauthorized update type at index");
```

Both functions also call the internal `_processUpdate()` function, so this validation can be de-duplicated by placing it there instead.

Chaos Labs: Fixed in commit 6cf09fb.

Cyfrin: Verified, the validation is now present in the shared internal function.

7.2.4 Parallel data structures are not necessary

Description: Usage of the `updatesById` mapping with keys given by the monotonically increasing `updateCounter` state variable is effectively the same as using the `updateHistory` array with an index shift of 1 (due to 0 being reserved for invalid update ids). In the current design, it is not necessary to maintain these parallel data structures, so if the format of update ids is unlikely to change in the future then this `updatesById` mapping can be removed in favor of the `updateHistory` array. Note that this modification would necessitate additional refactoring in the `getLatestUpdateByType()`, `getLatestUpdateByParameterAndMarket()`, and `getUpdateById()` functions.

Chaos Labs: Fixed in commit 6cf09fb.

Cyfrin: Verified, the `RiskParameterUpdate[] updateHistory` has been removed.

7.2.5 Unreachable code can be removed

Description: Within `RiskOracle::_processUpdate`, the `else` branch of the ternary operator is unreachable due to `updateCounter` being initialized to 0 and incremented before this line:

```
updateCounter++;
bytes memory previousValue = updateCounter > 0 ? updatesById[updateCounter - 1].newValue : bytes("");
```

Thus, this variable assignment logic can be simplified to just reading from the mapping (but note that this usage of the `updatedById` mapping is incorrect, as reported in M-01).

Chaos Labs: Fixed in commit 6cf09fb.

Cyfrin: Verified, the code path has been removed.

7.3 Gas Optimization

7.3.1 Unnecessary initialization can be removed

Description: Initialization of the `updateCounter` state variable within the constructor of `RiskOracle` is unnecessary and can be removed since this state will be `0` by default.

Chaos Labs: Fixed in commit 9f7375a.

Cyfrin: Verified, the initialization is no longer present.

7.3.2 Array length validation is not necessary

Description: `RiskOracle::publishBulkRiskParameterUpdates` currently validates that the lengths of all input arrays are equal.

```
function publishBulkRiskParameterUpdates(
    string[] memory referenceIds,
    bytes[] memory newValues,
    string[] memory updateTypes,
    bytes[] memory markets,
    bytes[] memory additionalData
) external onlyAuthorized {
    require(
        referenceIds.length == newValues.length && newValues.length == updateTypes.length
            && updateTypes.length == markets.length && markets.length == additionalData.length,
        "Mismatch between argument array lengths."
    );
    for (uint256 i = 0; i < referenceIds.length; i++) {
        require(validUpdateTypes[updateTypes[i]], "Unauthorized update type at index");
        _processUpdate(referenceIds[i], newValues[i], updateTypes[i], markets[i], additionalData[i]);
    }
}
```

This validation can be removed on account of the loop over `referenceIds`, as a length mismatch will either revert due to out-of-bounds access or result in additional elements beyond the length of the `referenceIds` array being ignored.

Chaos Labs: Fixed in commit 6cf09fb.

Cyfrin: Verified, the validation has been removed.