Clober LiquidityVault

Security Review

HickupHH3

22 January 2025

Contents

1 Introduction

The purpose of this audit is to review the Clober LiquidityVault contract, and its associated dependencies.

1.1 Audit Scope

The scope consisted of the `clober-rebalancer` repository in the `main` branch at commit hash `ef9902e79e0f624b245e33db2611f68f8b1d5189`. The repository name was subsequently renamed to `clober-liquidity-vault`. The contracts found in the `src` folder that were included in scope were the following:

File
Minter.sol
Operator.sol
Rebalancer.sol[1]
SimpleOracleStrategy.sol
interfaces/*
oracle/ChainlinkOracle.sol
oracle/DatastreamOracle.sol

[1] Rebalancer.sol was renamed to LiquidityVault.sol. The report will retain the old name as per the reviewed commit hash.

1.2 Audit Timeline

The audit was conducted from **13th January** to **17th January**.

1.3 Fix Review

A review of the fixes was conducted subsequently from **20th January** to **22nd January**.

1.4 Auditors Involved

HickupHH3

2 Risk Assessment Classification

There are 4 possible levels used to assess a vulnerability, with a separate section for gas optimizations.

High

Directly exploitable vulnerabilities with medium / high likelihood of loss of user funds, or contract functionality.

Resolving these issues are crucial to ensure the security and functionality of the contracts.

Medium

Vulnerabilities that relies on external dependencies / conditions to be met. Potentially leads to a loss of funds or functionality (eg. denial of service).

Resolving these issues are recommended to avoid undesired consequences.

Low

Issues arising from deviant behaviour than expected, but has no / little bearing from a security standpoint.

Informational

Issues that relate to security best practices recommendations, grammatical or styling errors, suggestions for variable/function name improvements etc. These issues are subjective and can be addressed based on the client's discretion.

While these issues may not directly affect the contract's functionality or security, addressing them can improve code readability, maintainability, and overall quality.

Gas Optimizations

Suggested changes to the codebase that will help reduce deployment or runtime gas costs, or to reduce the bytecode size should the limit be reached.

3 Findings Summary

Severity	No. of issues
High	0
Medium	0
Low	4
Informational	4
Gas Optimizations	0
Total	**8**

3.1 [Low] Datastream oracle must not exceed 5 price feeds

Context

DatastreamOracle.sol#L162-L171

Details

According to Chainlink's documentation, the permissible values for the `feeds` field is a maximum of 5 IDs.

However, `setFeed()` does not enforce a maxmimum cap when adding a new price feed. The theoretical maximum number of feeds supported by `DatastreamOracle` is 256, that is, `uint256 requestBitmap`, which exceeds Chainlink's permissible value.

Also, there isn't a method to prune stored `_feedIds`. While existing price feeds can have their `asset` modified, this may not be sufficient for price feed replacement.

Mitigation

Ensure that only up to 5 price feeds can be added. Consider adding a function to replace an existing price feed, though this comes with tradeoffs.

Response

Fixed at 777f41c.

Status

Fixed. The requests will be split into batches of 5 price feeds, with the bitmap updated for each batch request.

3.2 [Low] Sanity check that `fallbackOracle` has same decimals as `DatastreamOracle`

Context

DatastreamOracle.sol#L173-L175

Details

When setting a fallback oracle for the datasteamOracle, its decimals (precision) may differ, resulting in prices with differing precision returned.

Mitigation

Sanity check that the fallback oracle returns 18 decimals as well.

```
function setFallbackOracle(address newFallbackOracle) external onlyOwner {
+   if (fallbackOracle.decimals() != 18) revert DifferentPrecision();
    fallbackOracle = newFallbackOracle;
    emit SetFallbackOracle(newFallbackOracle);
}
```

Response

Fixed at 9dd01b0.

Status

Fixed.

3.3 [Low] `oraclePrice` calculation will revert for very large prices

Context

SimpleOracleStrategy.sol#L251

Details

For very large prices, there could be a risk of multiplication overflow when attempting to do `oraclePrice * 10 ** referenceOracle.decimals())`. This is illustrated in the POC below.

```
function testLargeOraclePriceRevertsFromMulOverflow() public {
    // revert from Panic error: integer overflow
    vm.expectRevert(stdError.arithmeticError);
    strategy.updatePosition(key, Tick.wrap(TickLibrary.MAX_TICK -
    10000).toPrice(), Tick.wrap(TickLibrary.MIN_TICK),
    Tick.wrap(TickLibrary.MIN_TICK), 1000000);
}
```

Mitigation

Use `Math.mulDiv()` for the calculation.

```
- oraclePrice = (oraclePrice * 10 ** referenceOracle.decimals()) >> 96;
+ oraclePrice = Math.mulDiv(oraclePrice, 10 ** referenceOracle.decimals(), 1
    << 96);
```

Response

Fixed at 260fad8.

Status

Fixed.

3.4 [Low] Large prcies exceeding `uint128` cannot be stored

Context

SimpleOracleStrategy.sol#L257

Details

After fixing the previous issue of multiplication overflow, another issue arises when attempting to safely downcast the oracle price to `uint128`, as it could theoretically exceed `type(uint128).max`. This is illustrated in the POC below.

```
function testLargeOraclePriceExceedsUint128() public {
    _setReferencePrices(1,
    uint256(58662020672688495886265712861148522827481680187947892) / 1e18);
    // @dev requires fixing oracle calc to use MulDiv first
    vm.expectRevert(
        abi.encodeWithSelector(
            SafeCast.SafeCastOverflowedUintDowncast.selector,
            128,
            uint256(293310103363442479431328564305742614137408400939733952)
        )
    );
    strategy.updatePosition(key, TickLibrary.MAX_PRICE - 1,
    Tick.wrap(TickLibrary.MIN_TICK), Tick.wrap(TickLibrary.MIN_TICK),
    1000000);
}
```

Mitigation

Change the `oraclePrice` type in `Position` to `uint176`. This allows for tight packing of the `Position` fields into a single word whilst accomodating the max price allowable by the Tick library.

```
struct Position {
    bool paused;
-   uint128 oraclePrice;
+   uint176 oraclePrice;
```

```
    uint24 rate;
    Tick tickA;
    Tick tickB;
}

- position.oraclePrice = SafeCast.toUint128(oraclePrice);
+ position.oraclePrice = SafeCast.toUint176(oraclePrice);
```

Response

Fixed at 9e475d9.

Status

Fixed.

3.5 [Info] Incomplete ERC6909 extension implementation

Context

Rebalancer.sol#L67-L69

Details

Looking at the ERC6909 metadata extension spec, the `name()`, `symbol()` and `decimals()` should be implemented, but `Rebalancer only implements the last method.

Mitigation

Either implement the `name()` and `symbol()` methods, or remove the `decimals()` funciton, as the metadata extension is optional.

Response

Fixed at 82845fa.

Status

Fixed. The `name()` and `symbol()` methods were implemented.

3.6 [Info] Use `forceApprove` instead of `approve`

Context

Minter.sol#L84

Details

`SafeERC20` is imported, but its `forceApprove` method is not used for approvals.

Mitigation

```
- IERC20(Currency.unwrap(currency)).approve(spender, amount);
+ IERC20(Currency.unwrap(currency)).forceApprove(spender, amount);
```

Response

Fixed at d1f5e5e.

Status

Fixed.

3.7 [Info] Consider making fee amount mutable

Context

Operator.sol#L48

Details

An immutable flat fee of 0.05 LINK is charged for calling `requestOraclePublic()`. Consider making this mutable to be able to respond to changing market conditions.

Response

Fixed at c9f9ee5.

Status

Fixed.

3.8 [Info] Additional sanity checks

Context

Rebalancer.sol#L58

Rebalancer.sol#L230-L233

Details

Consider adding the following checks:

- The immutable `burnFeeRate` set does not exceed `RATE_PRECISION`.
- `MIN_TICK.toPrice() < oraclePrice < MAX_TICK.toPrice()`

Mitigation

```
+ if (burnFeeRate_ >= RATE_PRECISION) revert InvalidConfig();
burnFeeRate = burnFeeRate_;

if (
+   oraclePrice < TickLibrary.MIN_PRICE ||
+   oraclePrice > TickLibrary.MAX_PRICE ||
    oraclePrice * (RATE_PRECISION + config.priceThresholdA) / RATE_PRECISION
        < priceA
    || oraclePrice * (RATE_PRECISION - config.priceThresholdB) /
        RATE_PRECISION > priceB
) revert ExceedsThreshold();
```

Response

Fixed at fcee5f1.

Status

Fixed.

4 Disclaimer

The audit report provided reflects a thorough review conducted to the best of my ability. However, it is important to note that the time-boxing nature of the review and available resources may prevent the discovery of all potential security vulnerabilities. As such, this audit does not guarantee the absence of undiscovered vulnerabilities.

Furthermore, please be aware that the security review was conducted on a specific commit of the codebase, as indicated. Any subsequent modifications made to the code will necessitate a new security review to ensure comprehensive coverage.

Note that the contracts used in production and expected deployment values may defer significantly from what was reviewed.

To ensure a robust evaluation of the codebase, it is highly recommended to engage multiple auditors and firms, particularly for large and complex projects. The involvement of multiple perspectives can provide additional insights and potential missed vulnerabilities.

Please consider these factors when assessing the audit report and making decisions related to the security and reliability of the smart contracts. The security review is not an endorsement of the project or its team, and should not be treated as such.