OlympusDAO

Security Review

HickupHH3

30 June 2023

Contents

1 Introduction

The purpose of this audit is to review the security of a new infrastructure system for the Olympus protocol: a generalized price oracle system to be used across Olympus products.

The first product to be migrated to use the new oracle system is the Range-Bound Stability (RBS) mechanism. Minor updates have been made to the RBS policies and are included in this audit. Additionally, a new primitive is introduced in the Default Framework to have additional abstraction needed for the oracle system: Submodules.

1.1 Audit Scope

The scope consisted of the `bophades` repository in the `price-v2` branch at commit hash `17fe660525b2f0d706ca318b53111fbf103949ba`. The contracts found in the `src` folder that were included in scope were the following:

File
Submodules.sol
policies/RBS/Heart.sol
policies/RBS/Operator.sol
policies/OCA/Bookkeeper.sol
modules/PRICE/PRICE.v2.sol
modules/PRICE/OlympusPrice.v2.sol
modules/PRICE/submodules/feeds/ChainlinkPriceFeeds.sol
modules/PRICE/submodules/feeds/UniswapV3Price.sol
modules/PRICE/submodules/feeds/UniswapV2PoolTokenPrice.sol
modules/PRICE/submodules/feeds/BalancerPoolTokenPrice.sol
modules/PRICE/submodules/strategies/SimplePriceFeedStrategy.sol

1.2 Audit Timeline

The audit was conducted from **19th June** to **30th June**.

1.3 Fix Review

A review of the fixes was conducted subsequently from **19th July** to **21st July**.

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	2
Medium	4
Low	5
Informational	9
Gas Optimizations	11
Total	31

3.1 [High] Incorrect constituent token price calculation for Balancer's stable pools

Context

BalancerPoolTokenPrice.sol#L807-L824

Details

There are a number of errors with the implementation of `BalancerPoolTokenPrice.getTokenPriceFromStablePool()`.

- The fetched vault balances need to be upscaled by the scaling factors. For instance, with the DOLA-USDC pool, attempting to fetch either USDC or DOLA will result in Arithmetic over/underflow error in `StableMath._calcOutGivenIn()`.
- Similarly, the calculated `lookupTokensPerDestinationToken` needs to be downscaled by the scaling factor because some pools' scaling factors factor in the token rate, like for appreciating assets against the base token. An example of this is the rETH-WETH pool, where fetching the rETH price incorrectly returned an approximately equivalent WETH price.
- A puzzling discovery was made with the DOLA-USDC pool, where the invariant had to be calcalated instead of using the fetched `invariant` parameter. Theoretically they should have been the same; the cause of the issue is unknown.

Impact

These errors would have led to incorrect prices calculated by the price feed adapter.

Mitigation

After much tinkering, the implementation below seems to return accurate values.

```
uint256 lookupTokensPerDestinationToken;
{
  (, uint256[] memory balances_, ) = balVault.getPoolTokens(poolId);
  try pool.getLastInvariant() returns (
    uint256, // invariant,
    uint256 ampFactor
  ) {
    // Fix#1: upscale amounts by scaling factors
    uint256[] memory scalingFactors = pool.getScalingFactors();
    for (uint i; i < scalingFactors.length; ++i) {
      balances_[i] = FixedPoint.mulDown(balances_[i], scalingFactors[i]);
    }

    // Calculate the quantity of lookupTokens returned by swapping 1 destinationToken
    lookupTokensPerDestinationToken = StableMath._calcOutGivenIn(
      ampFactor,
      balances_,
      destinationTokenIndex,
      lookupTokenIndex,
      1e18,
      // Fix#2: invariant calculation needed instead of
      // returned invariant param
      StableMath._calculateInvariant(ampFactor, balances_)
    );

    // Fix#3: downscale amount to token decimals
    lookupTokensPerDestinationToken = FixedPoint.divDown(
      lookupTokensPerDestinationToken,
      scalingFactors[lookupTokenIndex]
    );
  } catch (bytes memory) {
    // Revert if the pool is not a stable pool, and does not have the required function
    revert Balancer_PoolTypeNotStable(poolId);
  }
}

// value is in lookup decimals, convert to outputDecimals_
lookupTokensPerDestinationToken = _convertERC20Decimals(
  lookupTokensPerDestinationToken,
  lookupToken_,
  outputDecimals_
  );

// Price per destinationToken / quantity
return destinationTokenPrice.mulDiv(
  10 ** outputDecimals_,
  lookupTokensPerDestinationToken
);
```

Response

Fixed in PR #160.

Status

Fixed. Recommendation was adopted with minor differences.

3.2 [High] Incorrect Balancer stable token BPT calculation

Context

BalancerPoolTokenPrice.sol#L518-L528

Details

Balancer's documentation regarding BPT valuation has become unclear on what the `getRate()` function is relative to, with merely a couple of examples given.

To determine the BPT pool value `getStablePoolTokenPrice()`, the implementation fetches the pool's rate via `pool.getRate()`, then multiplies it with the price of the first token of the pool.

```
address token = tokens[0];
if (token == address(0)) revert Balancer_PoolTokenInvalid(poolId, 0, token);
(uint256 price_, ) = _PRICE().getPrice(token, PRICEv2.Variant.CURRENT); // outputDecimals_
baseTokenPrice = price_;
uint256 poolValue = poolRate.mulDiv(baseTokenPrice, 10 ** poolDecimals); //
↪  outputDecimals_
```

This is erroneous, as the now-deleted document on valuing BPT as collateral] states it should take the minimum price of all constituent tokens.

Furthermore, in the docs had the following note before removal:

> Note that the method above does not account for price divergence from the assumed peg. If `stETH` depegs from `ETH`

or any stablecoin depegs from USD, `pool.getRate()` will suffer inaccuracies.

It is unclear whether the inaccuracy still holds.

Regardless, we can verify that the minimum of the prices should indeed be used, as per the example below.

POC

Consider the rETH-WETH pool. At last change block `17581786`:

- vault.getPoolTokens: `[rETH, WETH]`
- rETH balance = `22052747029264318843176` (22052.747)
- WETH balance = `2394120155977342766942 3` (23941.202)
- pool.getRate(): `1023539846410394940` (1.02354)
- WETH price: $1838
- rETH price: $1979
- BPT supply: `46556783987856123852040` (46556.784)

Expected BPT value

1 BPT = ((rETH balance * rETH price) + (WETH balance * WETH price)) / BPT supply = `(22052.747 * 1979 + 23941.202 * 1838) / 46556.784 = 1882.56`

Calculated BPT value

Since the 1st token is rETH, assuming oracle decimals of 18, 1 BPT = `1023539846410394940 * 1979e18 / 10**18 = 2.0255853560461714e+21 = 2025.59`

Hence, we see an over-valuation of the pool BPT token.

If we were to use the minimum token price (WETH) instead, it would be `1023539846410394940 * 1838e18 / 10**18 = 1.8812662377023057e+21 = 1881.27` which is much closer to the expected value.

Impact

Over-valution of BPT pool token.

Mitigation

Compute and use the minimum price of all constituent tokens instead of just the first consitutent token.

Also, as per the removed comment, it is unclear whether the `pool.getRate()` function will accurately price in consitutent or underlying token depegs. It is thus recommended to caution the use of this `getStablePoolTokenPrice()` function should depegs occur.

Response

Fixed in PR #162.

Status

Fixed. Minimum price of constituents is determined and used instead of only the first consitutent. Comments were also added to warn of inaccurate prices should there de-pegs occur.

3.3 [Medium] `BalancerPoolTokenPrice` incompatibility with Balancer's Composable Stable Pools

Context

BalancerPoolTokenPrice.sol#L809

Details

According to Balancer's docs, Composable Stable Pools are a superset of all previous Stable-type pools (Stable Pools, MetaStable Pools, StablePhantom Pools, and StablePool v2) and therefore deprecate all previous pools.

This pool type seems to lack the `getLastInvariant()` API that its predecessors have, which the `BalancerPoolTokenPrice` contract relies on for calculation

of a token's price in the pool. Hence, it'll be unable to add such pools as price feeds.

Impact

Composable stable pools cannot be added as price feeds. Notable composable stable pools with substantial liquidity include the R-DAI, wstETH-sfrxETH-rETH and Balancer Boosted Aave V3 USD pools.

Mitigation

From rather extensive research, it seems that a separate adapter function would be required, as Composable Stable pools have another property of pre-minted BPT that futher complicates the calculation where:

1. The max `2**(111)` BPT is minted upon creation, so the virtual supply has to be fetched instead of the actual supply.
2. The BPT token is itself included as one of the tokens in the pool, which would be problematic in calculating the LP token price because of its recursive nature in fetching the constituent prices from the parent PRICE module.

Response

Fixed in PR #172. The remediation in this PR is to check if the pool passed in via parameters has the `getLastVariant()` function (which composable stable pools lack).

Status

Fixed. Comment has been added to note the incompatibility of composable stable pools.

3.4 [Medium] Duplicate price feed components can be added

Context

OlympusPrice.v2.sol#L467-L478

Details

There isn't a check to ensure that the price feed components are distinct. It is however a feature to allow the same price feed and selector (Eg. taking prices from multiple UniV2 pools).

Impact

Duplicate price feeds will skew the average / median price calculated by the `SimplePriceFeedStrategy`.

Mitigation

Hash the component feed (target, selector & params) and check against that for duplicates. Have the feeds sorted by the hashes in ascending order so that they need not be SSTOREd / SLOADed.

```
bytes32 prevComponentHash = keccak256(
  abi.encodePacked(
    feeds_[0].target,
    feeds_[0].selector,
    feeds_[0].params
  )
);

for (uint256 i; i < len; ) {
  if (!_submoduleIsInstalled(feeds_[i].target))
    revert PRICE_SubmoduleNotInstalled(asset_, abi.encode(feeds_[i].target));
    if (i != 0) {
      bytes32 currentComponentHash = keccak256(
        abi.encodePacked(
          feeds_[i].target,
          feeds_[i].selector,
          feeds_[i].params
        )
      );

      if (prevComponentHash >= currentComponentHash) revert DuplicateComponent();

      prevComponentHash = currentComponentHash;
    }

    unchecked {
        ++i;
    }
}
```

Response

Fixed in PR #165.

Status

Fixed. Implemented inner for loop for duplicate checks.

3.5 [Medium] Reduced likelihood of average / median price returned due to incorrect divisor in deviation check

Context

SimplePriceFeedStrategy.sol#L192
SimplePriceFeedStrategy.sol#L250

Details

The deviation check of the maximum price from the average is as follows:

```
// Check the deviation of the maximum from the average
uint256 maxPrice = sortedPrices[sortedPrices.length - 1];
if (((maxPrice - averagePrice) * 10000) / maxPrice > deviationBps) return averagePrice;
```

Because the divisor is `maxPrice` instead of `averagePrice`, the calculated deviation would be less than expected.

Example:

- `maxPrice = 15e17` (1.5 ETH)
- `averagePrice = 1e18` (1 ETH)
- expected deviation = 50% = 5000 bps
- actual deviation = `(15e17 - 1e18) / 15e17` = 33.3% = 3333 bps

Impact

The lower calculated deviation therefore means a lower likelihood of the function returning the average / median price. This results in a higher probability of the first value returned.

Mitigation

Replace `maxPrice` with `averagePrice` as the divisor.

```
- if (((maxPrice - averagePrice) * 10000) / maxPrice > deviationBps) return averagePrice;
+ if (((maxPrice - averagePrice) * 10000) / averagePrice > deviationBps) return
↪  averagePrice;
- if (((maxPrice - averagePrice) * 10000) / maxPrice > deviationBps) return medianPrice;
+ if (((maxPrice - averagePrice) * 10000) / averagePrice > deviationBps) return
↪  medianPrice;
```

Response

Fixed in PR #167.

Status

Fixed. In addition, `deviationBps` is checked to be `0 < deviationBps < DEVIATION_MAX`.

3.6 [Medium] Unsafe uint128 casting if `39 <= baseTokenDecimals <= BASE_10_MAX_EXPONENT = 50`

Context

UniswapV3Price.sol#L228

Details

`baseTokenDecimals` is checked to not exceed `BASE_10_MAX_EXPONENT` to prevent overflow issues. The base amount passed into `getQuoteAtTick()` is `uint128(10 ** baseTokenDecimals)`.

`type(uint128).max` is of magnitude 1e38. Hence, there will be an unsafe casting to a much smaller number if the token decimals is between 39 to 50.

Impact

The `baseInQuotePrice` and subsequent `tokenPrice` will be much smaller than expected.

Mitigation

The largest token decimals I've seen from experience is 24, so a lower decimal limit `BASE_10_MAX_EXPONENT` to about 30 should suffice.

Response

Fixed in PR #168.

Status

Fixed.

3.7 [Low] `{PARENT}.` is treated to be a valid subkeycode

Context

Submodules.sol#L39-L49

Details

The rules of the subkeycode naming are the following:

- Must have the parent Keycode as its first 5 bytes
- Only contain A-Z, _, or blank characters
- 6th byte must be a period "."

With these, a valid subkeycode is to have all null characters after the period, eg. (`PRICE.`), which arguably shouldn't be valid.

POC

Replace the `SUBKEYCODE()` of `MockInvalidSubKeycodeThree` to `MOCKY`.

```
function SUBKEYCODE() public pure override returns (SubKeycode) {
-   return toSubKeycode("MOCKY.SUBMODULE!");
+   return toSubKeycode("MOCKY.");
}
```

Then, the `testRevert_installSubmodule_invalidSubKeycode()` would not revert as expected, and `testRevert_upgradeSubmodule_invalidSubKeycode()` would revert for attempting to upgrade the same module (different revert reason than expected).

Mitigation

The 6th character should be specially handled as well (shouldn't be null).

Response

Fixed in PR #166.

Status

Fixed.

3.8 [Low] Incorrect revert error if requested observation window exceeds oldest observation

Context

UniswapV3Price.sol#L175-L186

Details

The external call to `pool.observe()` reverts if `observationWindowSeconds` exceeds the pool's oldest observation. This will be caught in the `catch()` clause, which incorrectly reverts with the `UniswapV3_PoolTypeInvalid` error.

Mitigation

For simplicity, consider renaming the error to accomodate this revert.

Response

Fixed in PR #169.

Status

Fixed. The check for UniswapV3 pools has been changed from `token0` to `slot0`, with a try-catch for the `pool.observe()` method that exits with the `UniswapV3_InvalidObservation` custom error should it revert.

3.9 [Low] 1st (possibly zero) median price returned instead of 1st non-zero price

Context

SimplePriceFeedStrategy.sol#L208
SimplePriceFeedStrategy.sol#L253

Details

If no deviation is detected, the 1st non-zero price in the array should be returned.

However, the median price implementation returns the first value of the `prices_` array, which may be zero.

Impact

Increased likelihood of 0 median price returned (1st element vs all elements being 0).

Mitigation

The `getAveragePriceIfDeviation()`'s implementation is correct: taking the 1st value from the `nonZeroPrices` array.

```
// Otherwise, return the first value
- return prices_[0];
+ return nonZeroPrices[0];
```

Response

Fixed in PR #173.

Status

Fixed. `nonZeroPrices[0]` is cached in `firstNonZeroPrice` and returned.

3.10 [Low] Minor differences with canonical UniswapV3 libraries

Context

OracleLibrary.sol
TickMath.sol

Details

The libraries used have minor differences from the `0.8` branch of the UniswapV3 core and peripheral repos.

Mitigation

Use the canonical OracleLibrary and TickMath libraries that have been modified to be compatible with solc `^0.8`.

Response

Fixed in PR #169.

Status

Fixed. Differences are a result of import file paths and linting.

3.11 [Low] Cautionary use of TWAPs on Ethereum mainnet

Context

UniswapV3Price.sol#L133-L247

Details

The team is aware that using UniV2's spot price feed for underlying tokens is susceptible to flash loan manipulation, and should be used with caution.

Similarly, TWAPs (UniV2 UniV3) are also susceptible to multi-block MEV manipulation that is theoretically possible on Ethereum mainnet, as detailed in this ChainSecurity article.

Mitigation

It is advisable to caution the use of `UniswapV3Price.getTokenTWAP()` as well.

Response

Fixed in commit ddd065f.

Status

Fixed.

3.12 [Info] Important assumption that `outputDecimals_` is equal to `PRICE.decimals()`

Details

In the price feed adapters, there is an important assumption made when recursive calls to the PRICE parent module is made: `PRICE.decimals()` is equal to the input `outputDecimals_` parameter.

This may not necessarily be so (Eg. if the adapters are called by external 3rd parties), resulting in incorrect precision of the returned price.

Mitigation

Consider enforcing `outputDecimals_` being equal to `PRICE.decimals()`.

Response

Acknowledged. Adapters are not expected to be used individually by third-parties.

Status

Acknowledged.

3.13 [Info] Add adapters for Balancer RateProviders

Details

Balancer has rate providers that provide exchange rates for some assets. Some instances are:

- WstETHRateProvider
- CbEthRateProvider

These providers have a simple `getRate()` function that unfortunately isn't compatible with any of the price feed contracts implementations.

Mitigation

Consider adding a price feed adapter for Balancer's rate providers.

Response

Acknowledged. Out of scope, as they won't be used.

Status

Acknowledged.

3.14 [Info] Inline comments of `Submodules.ensureValidSubKeycode()` can be improved

Context

Submodules.sol#L30-L31
Submodules.sol#L48

Details

- Inconsistency between L31 and L48: L31 doesn't mention numbers 0-9 being valid
- The commented lines L30 & L31 describe the criteria for a valid subkeycode, but omitted 1 criteria rule: the 6th character has to be a period

Mitigation

- The mentioned range of ASCII characters should be consistent
- Add the missing rule

Response

Fixed in PR #166.

Status

Fixed. A new rule was added too: the portion after the period must start with 3 non-blank characters.

3.15 [Info] Broken Balancer URL links

Context

BalancerPoolTokenPrice.sol#L358
BalancerPoolTokenPrice.sol#L451

Details

The referenced URL links lead to 404 of the balancer docs due to recent changes.

Mitigation

Update the links to

`https://docs.balancer.fi/concepts/advanced/valuing-bpt/valuing-bpt.html#on-chain-price-evaluation`.

Response

Fixed in commit 79d4c63.

Status

Fixed.

3.16 [Info] Natspec & Comment Improvements

Context

SimplePriceFeedStrategy.sol#L148
SimplePriceFeedStrategy.sol#L203
SimplePriceFeedStrategy.sol#L264
SimplePriceFeedStrategy.sol#L289

SimplePriceFeedStrategy.sol#L172
SimplePriceFeedStrategy.sol#L227
SimplePriceFeedStrategy.sol#L258

Details

- Incorrect comment on non-zero prices being ignored (should be zero prices)
- `If there are no non-zero prices in the array` is a double negation and requires a slightly higher mental load to process.

Mitigation

```
- Non-zero prices in the array are ignored
+ Zero prices in the array are ignored

- If there are no non-zero prices in the array, 0 will be returned.
+ Return 0 if all prices in the array are zero.
```

Response

Fixed in commit b50152e.

Status

Fixed.

3.17 [Info] Function Naming Inconsistency

Context

Bookkeeper.sol#L72
Bookkeeper.sol#L96

Details

All function names in the Bookkeeper calling the `PRICE` module use the same underlying names except for `addAssetPrice()` and `removeAssetPrice()`.

Mitigation

Rename the referenced functions to maintain consistency of the naming style.

```
- function addAssetPrice(
+ function addAsset(

- function removeAssetPrice(address asset_) external onlyRole("bookkeeper_policy") {
+ function removeAsset(address asset_) external onlyRole("bookkeeper_policy") {
```

Response

With TRSRYv1.1 we add assets and categorize them in that contract too, so `addAssetPrice` will be different from addAsset on the Bookkeeper.

Status

Acknowledged.

3.18 [Info] Renaming Bookkeeper

Context

Bookkeeper.sol

Details

The purpose of the Bookkeeper Policy is to "provide access-controlled functions for managing PRICEv2 module data configuration and submodules".

Its name isn't suggestive of its intended purpose, as bookkeeping is an accounting term: "the process of recording your company's financial transactions into organized accounts on a daily basis".

Furthermore, there could be other similar policies to handle the management of submodules in the future.

Mitigation

Consider renaming `Bookkeeper` something to the effect of managing submodules, like `PriceSubmoduleManager`.

Response

Acknowledged. Going to keep as-is because its functionality will be expanded in the future.

Status

Acknowledged.

3.19 [Info] Spelling Error

Context

UniswapV2PoolTokenPrice.sol#L178

Mitigation

```
- succeptibility
+ susceptibility
```

In addition, the VSC Code Spell Checker Extension helps to spot any spelling errors. Recommend installing it.

Response

Fixed in commit c7b1073.

Status

Fixed.

3.20 [Info] Balancer's StableMath library comments on unchecked arithmetic should be ignored

Context

StableMath.sol

Details

Similar to UniswapV3, the StableMath library is written in solc `^0.7`, and has some comments regarding the use of unchecked arithmetic. Based on this GH comment and GH issue, it seems that the pragma version can be simply changed to 0.8 and above without futher modifications to the library, and that the library comments are outdated.

Mitigation

Noteworthy, but no remediation is required.

Response

N.A.

Status

N.A.

3.21 [Gas] Simplified `UniswapV2PoolTokenPrice.getTokenPrice()` implementation

Context

UniswapV2PoolTokenPrice.sol#L334-L369

Details

The referenced lines are a little complicated and can be simplified to just 6 lines of code.

Mitigation

The referenced lines can be converted to the following:

```
if (tokens_[0] == address(0)) revert UniswapV2_PoolTokensInvalid(address(pool), 0,
↪  tokens_[0]);
if (tokens_[1] == address(0)) revert UniswapV2_PoolTokensInvalid(address(pool), 1,
↪  tokens_[1]);
if (lookupToken_ != tokens_[0] && lookupToken_ != tokens_[1]) revert
↪  UniswapV2_LookupTokenNotFound(address(pool), lookupToken_);
uint256 lookupTokenIndex = (lookupToken_ == tokens_[0]) ? 0 : 1;
uint256 destinationTokenIndex = 1 - lookupTokenIndex;
(uint256 destinationTokenPrice, ) = _PRICE().getPrice(tokens_[destinationTokenIndex],
↪  PRICEv2.Variant.CURRENT);
```

Response

Fixed in commit dbab871.

Status

Fixed.

3.22 [Gas] Cache fetched Balancer pool params into a struct to avoid repeated external calls

Context

BalancerPoolTokenPrice.sol#L556-L696
BalancerPoolTokenPrice.sol#L245-L258

Details

There are repeated external calls to the balancer pool to retrieve the token addresses, balances, weights, decimals etc, which can be rather costly.

By caching these results into a struct memory, the external calls can be made just once, thereby seeing significant gas savings.

Mitigation

The implementation below introduces a `BalancerWeightedPoolCache` struct that caches the external calls made to the balancer pool while avoiding the stack too deep problem. About 20k gas savings were observed.

```
test_getWeightedPoolTokenPrice_tokenDecimalsFuzz(uint8) (gas: -69 (-0.034%))
test_getTokenPriceFromWeightedPool_incorrectPoolType() (gas: 187 (0.091%))
test_getTokenPriceFromWeightedPool_threeTokens_twoBalances_threeWeights() (gas: 233 (0.134%))
test_getTokenPriceFromWeightedPool_twoTokens_threeBalances_threeWeights() (gas: 233 (0.134%))
test_getTokenPriceFromWeightedPool_threeTokens_threeBalances_twoWeights() (gas: 251 (0.145%))
test_getWeightedPoolTokenPrice_poolTokenDecimalsFuzz(uint8) (gas: -372 (-0.175%))
test_getStablePoolTokenPrice_poolTokenDecimalsFuzz(uint8) (gas: 205 (0.195%))
test_getTokenPriceFromWeightedPool_noPrice() (gas: 209 (0.336%))
test_getTokenPriceFromWeightedPool_unknownToken() (gas: 222 (0.353%))
test_getTokenPriceFromWeightedPool_coinTwoZero() (gas: 226 (0.382%))
test_getTokenPriceFromWeightedPool_coinOneZero() (gas: 217 (0.391%))
test_getTokenPriceFromWeightedPool_poolDecimalsMaximum() (gas: 197 (0.391%))
test_getTokenPriceFromStablePool_priceDecimalsFuzz(uint8) (gas: -361 (-0.423%))
test_getTokenPriceFromWeightedPool_threeTokens() (gas: -20213 (-8.532%))
test_getTokenPriceFromWeightedPool_fuzz(uint8,uint8,uint8) (gas: -18118 (-12.680%))
test_getTokenPriceFromWeightedPool_poolDecimalsFuzz(uint8) (gas: -18118 (-14.941%))
test_getTokenPriceFromWeightedPool_inverse() (gas: -18118 (-15.973%))
test_getTokenPriceFromWeightedPool_tokenDecimalsMaximum() (gas: -15066 (-16.591%))
test_getTokenPriceFromWeightedPool_tokenDecimalsFuzz(uint8) (gas: -18118 (-17.362%))
test_getTokenPriceFromWeightedPool_priceDecimalsFuzz(uint8) (gas: -18118 (-17.687%))
test_getTokenPriceFromWeightedPool_priceDecimalsSame() (gas: -18118 (-20.256%))
Overall gas change: -142609 (-0.017%)
```

```
struct BalancerWeightedPoolCache {
  address[] tokens;
  uint256[] weights;
  uint256[] balances;
  uint8 decimals;
  bytes32 poolId;
}

function _getTokenBalanceWeighting(
  BalancerWeightedPoolCache memory cache,
  uint256 index,
  uint8 outputDecimals_
) internal view returns (uint256) {
  uint256 tokenBalance = _convertERC20Decimals(
    cache.balances[index],
    cache.tokens[index],
    outputDecimals_
    );

  uint256 tokenWeight = cache.weights[index].mulDiv(10 ** outputDecimals_, 10 **
↪  cache.decimals);

  return tokenBalance.mulDiv(10 ** outputDecimals_, tokenWeight);
}

function getTokenPriceFromWeightedPool(
  address lookupToken_,
  uint8 outputDecimals_,
  bytes calldata params_
) external view returns (uint256) {
```

```
  // Prevent overflow
  if (outputDecimals_ > BASE_10_MAX_EXPONENT)
      revert Balancer_OutputDecimalsOutOfBounds(outputDecimals_, BASE_10_MAX_EXPONENT);

  // Decode params
  IWeightedPool pool;
  {
    BalancerWeightedPoolParams memory params = abi.decode(
      params_,
      (BalancerWeightedPoolParams)
    );
    if (address(params.pool) == address(0)) revert
↪   Balancer_PoolTypeNotWeighted(bytes32(0));

    pool = IWeightedPool(params.pool);
  }

  BalancerWeightedPoolCache memory poolCache;
  uint256 lookupTokenIndex = type(uint256).max;
  uint256 destinationTokenIndex = type(uint256).max;
  uint256 destinationTokenPrice; // Scale: outputDecimals_
  {
    // Prevent re-entrancy attacks
    VaultReentrancyLib.ensureNotInVaultContext(balVault);

    // Get tokens in the pool from vault
    poolCache.poolId = pool.getPoolId();
    (address[] memory tokens_, uint256[] memory balances_, ) = balVault.getPoolTokens(
      poolCache.poolId
    );

    poolCache.tokens = tokens_;
    poolCache.balances = balances_;

    poolCache.decimals = pool.decimals();
    if (poolCache.decimals > BASE_10_MAX_EXPONENT)
      revert Balancer_PoolDecimalsOutOfBounds(
        poolCache.poolId,
          poolCache.decimals,
          BASE_10_MAX_EXPONENT
        );

      // Test if the weights function exists, otherwise revert
      try pool.getNormalizedWeights() returns (uint256[] memory weights_) {
        poolCache.weights = weights_;
      } catch (bytes memory) {
        revert Balancer_PoolTypeNotWeighted(poolCache.poolId);
      }

      if (!(poolCache.tokens.length == poolCache.balances.length &&
↪   poolCache.balances.length == poolCache.weights.length))
```

```
        revert Balancer_PoolTokenBalanceWeightMismatch(
          poolCache.poolId,
          poolCache.tokens.length,
          poolCache.balances.length,
          poolCache.weights.length
        );

      // Determine the index of the lookup token and an appropriate destination token
      uint256 tokensLen = poolCache.tokens.length;
      for (uint256 i; i < tokensLen; i++) {
        // If address is zero, complain
        address currentToken = poolCache.tokens[i];
        if (currentToken == address(0))
          revert Balancer_PoolTokenInvalid(poolCache.poolId, i, currentToken);

        // If lookup token
        if (lookupToken_ == currentToken) {
          lookupTokenIndex = i;
          continue;
        }

        // Don't set the destination token again
        if (destinationTokenIndex != type(uint256).max) continue;

        /**
         * PRICE will revert if there is an issue resolving the price, or if it is 0.
         *
         * We catch this, so that other candidate tokens can be tested.
         */
        try _PRICE().getPrice(currentToken, PRICEv2.Variant.CURRENT) returns (
          uint256 currentPrice,
          uint48 // timestamp
        ) {
          destinationTokenIndex = i;
          destinationTokenPrice = currentPrice;
        } catch (bytes memory) {
            continue;
        }
      }

      // Lookup token not found
      if (lookupTokenIndex == type(uint256).max)
        revert Balancer_LookupTokenNotFound(poolCache.poolId, lookupToken_);

      // No destination token found with a price
      if (
        destinationTokenPrice == 0 ||
        destinationTokenIndex == type(uint256).max
      ) revert Balancer_PriceNotFound(poolCache.poolId, lookupToken_);
    }
```

```
  // Calculate the rate of the lookup token
  uint256 lookupTokenUsdPrice;
  {
    // Weightings
    // Scale: outputDecimals_
    uint256 lookupTokenWeighting = _getTokenBalanceWeighting(
      poolCache,
      lookupTokenIndex,
      outputDecimals_
    );

    uint256 destinationTokenWeighting = _getTokenBalanceWeighting(
      poolCache,
      destinationTokenIndex,
      outputDecimals_
    );

    // Get the lookupToken in terms of the destinationToken
    // Source: https://docs.balancer.fi/reference/math/weighted-math.html#spot-price
    lookupTokenUsdPrice = destinationTokenWeighting.mulDiv(
      destinationTokenPrice,
      lookupTokenWeighting
    );
  }
  return lookupTokenUsdPrice;
}
```

Response

Fixed in commit 149b6f2.

Status

Fixed.

3.23 [Gas] Cache repeated external calls

Context

ChainlinkPriceFeeds.sol#L214-L225
ChainlinkPriceFeeds.sol#L260-L283

ChainlinkPriceFeeds.sol#L321-L344

UniswapV3Price.sol#L159-L164

Details

Similar to the finding above this, there are instances of repeated external calls that can be cached.

- `params.feed.decimals()` which is called again in `_getFeedPrice()`
- `pool.token0()` & `pool.token1()`

Mitigation

Cache the external calls into memory.

Response

Fixed in commit 1868177.

Status

Fixed.

3.24 [Gas] Cache decimals variable into memory when used in for-loop

Context

OlympusPrice.v2.sol#L138

Details

When iterating through the price feeds, the `decimals` storage variable is used. It's more gas efficient to store its value into memory and use that.

Mitigation

```
+ uint8 _decimals = decimals;
for (uint256 i; i < numFeeds; ) {
  (bool success_, bytes memory data_) = address(_getSubmoduleIfInstalled(feeds[i].target))
    .staticcall(
-       abi.encodeWithSelector(feeds[i].selector, asset_, decimals, feeds[i].params)
+       abi.encodeWithSelector(feeds[i].selector, asset_, _decimals, feeds[i].params)
  );
```

Response

Fixed in commit bdfb902.

Status

Fixed.

3.25 [Gas] Redundant `destinationToken`

Context

BalancerPoolTokenPrice.sol#L739
BalancerPoolTokenPrice.sol#L801

Details

The `destinationToken` is redundant as its functionality is covered by the `destinationTokenIndex`.

Mitigation

Remove the referenced lines.

Response

Fixed in commit aeac949.

Status

Fixed.

3.26 [Gas] Redundant else case for enum Variant check

Context

OlympusPrice.v2.sol#L108-L110

Details

The compiler handles enum variant validation, so the else case can never be reached and is therefore redundant.

Mitigation

Remove the else case.

Response

Fixed in commit 1760532.

Status

Fixed.

3.27 [Gas] Duplicate price length checks

Context

SimplePriceFeedStrategy.sol#L97-L101
SimplePriceFeedStrategy.sol#L227-L231
SimplePriceFeedStrategy.sol#L313-L314

Details

The price length checks of 0 and 1 in `_getMedianPrice()` is redundant because the length of the array has been guaranteed to be at least 3 before its invocation.

Mitigation

Either the price length checks in `_getMedianPrice()` can be removed (and add the pre-condition that it should've been checked as a @dev note), or remove the checks in its parent functions .

Response

Fixed in commit e744a53.

Status

Fixed. Price length checks in `_getMedianPrice()` were removed.

3.28 [Gas] Redundant subtraction for odd price length

Context

SimplePriceFeedStrategy.sol#L111

Details

As solidity division rounds down, the subtraction of priceLen by 1 for odd-length price lengths is redundant.

Mitigation

```
- return prices_[(pricesLen - 1) / 2];
+ return prices_[pricesLen / 2];
```

Response

Fixed in commit 8128afd.

Status

Fixed.

3.29 [Gas] Redundant castings

Context

UniswapV3Price.sol#L181
Heart.sol#L205

Details

1. The `int56` casting of `params.observationWindowSeconds` is redundant since the numerator and result are `int56`. While it can be argued that it is unsafely casted from `uint32` to `int32`, it is very likely that it'll revert when called in `pool.observe(observationWindow)` because of the extremely large observation window for unsafe castings.

2. Similarly, `duration` need not be casted to `uint256` in `currentReward()`.

Mitigation

```
- int56(int32(params.observationWindowSeconds));
+ int32(params.observationWindowSeconds);

- (uint256(currentTime - nextBeat) * maxReward) / uint256(duration);
+ (uint256(currentTime - nextBeat) * maxReward) / duration;
```

Response

Fixed in commit 3ebbca7.

Status

Fixed.

3.30 [Gas] Redundant config param check

Context

Operator.sol#L107

Details

The `configParams[7] == uint32(0)` check is redundant because it's indirectly checked by the previous and next conditions following it.

```
if (
  configParams[5] < 1 hours ||
  configParams[6] > configParams[7] || // condition #1
  configParams[7] == uint32(0) ||
  configParams[6] == uint32(0) // condition #2
) revert Operator_InvalidParams();
```

As a result of conditions #1 & #2, we have `0 < configParams[7]`. Hence, `configParams[7]` cannot be zero.

Mitigation

The referenced line can be removed.

Response

Fixed in commit 736962e.

Status

Fixed.

3.31 [Gas] Redundant imports and event

Context

OlympusPrice.v2.sol#L4
UniswapV3Price.sol#L9

Bookkeeper.sol#L20

Mitigation

The referenced lines are unused and can be removed.

Response

Fixed in commit 77442d7.

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.