OpenZeppelin | security

Origin Balancer MetaPool Strategy Audit

ORIGIN

September 20, 2023

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-08-07 To 2023-08-16
Languages	Solidity

Total Issues	19 (14 resolved, 2 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	3 (3 resolved)
Medium Severity Issues	4 (2 resolved)
Low Severity Issues	4 (3 resolved)
Notes & Additional Information	6 (5 resolved, 1 partially resolved)
Client Reported Issues	2 (1 resolved, 1 partially resolved)

Scope

We audited the OriginProtocol/origin-dollar repository at the `eb11498c376b65696c90981757221b076d6226aa` commit.

In scope were the following contracts:

```
contracts
└── strategies
    └── balancer
        ├── BalancerMetaPoolStrategy.sol
        ├── BaseAuraStrategy.sol
        └── BaseBalancerStrategy.sol
```

System Overview

Origin Ether (OETH) is an ERC-20-compliant token collateralized by Rocket Pool ETH, Lido Staked ETH, Frax ETH and Wrapped ETH. Users can mint OETH by depositing any of these assets in the system's vault to later be invested into different strategies, such as Curve pools.

The audited commit introduces the Balancer MetaPool strategy. This new strategy for generating yield takes advantage of the Aura Finance protocol to boost the yield on collateral deposits to a Balancer MetaPool.

When depositing collateral to Balancer pools, the strategy receives LP tokens called Balancer Pool Tokens (BPTs), which represent the share of the liquidity pool that is provided by the strategy. On top of earning trading fees, Balancer LPs are also rewarded with BAL tokens, proportionally to the amount of liquidity they provided. In order to boost yield, the `BalancerMetaPoolStrategy` deposits the BPTs to Aura, which results in earning more BAL rewards, as well as AURA tokens. Naturally, the process is reversed when the strategy needs to withdraw collateral to the vault.

The `BaseBalancerStrategy` contract implements generic logic around Balancer pools, while the `BaseAuraStrategy` contract adds logic to deposit and withdraw BPTs to Aura pools, as well as collect rewards.

The `BalancerMetaPoolStrategy` contract implements logic specific to Balancer MetaPools, which are pools geared towards highly correlated but not completely equivalent assets, like the ones used as collateral by the Vault (`wEth`, `rETH`, `stETH`, and `frxETH`).

To make the strategy compatible with the OETH Vault, the `BaseBalancerStrategy` contract inherits from `InitializableAbstractStrategy`.

While the strategy supports rebasing tokens such as `stEth` and `frxEth`, Balancer pools do not, so additional wrapping and unwrapping logic is required when depositing and withdrawing liquidity from Balancer. Note that it is intended for the strategy to not necessarily support all pool assets, and it is expected to only deposit/withdraw a subset of those. We will use "strategy assets" for the assets supported by the strategy, and "pool assets" for the assets supported by the pool, in their wrapped versions.

Note that the strategy was reviewed with Balancer MetaStable pools in mind, and the current shared functionality such as pricing the BPT tokens or estimating the strategy's underlying value may need to be updated when working with other types of Balancer pools.

Security Model and Trust Assumptions

Additional assumptions as to the previous audits are as follows:

- The `BalancerErrors` and `VaultReentrancyLib` libraries are part of Balancer's codebase, and as such they are out of scope for this audit and are expected to work correctly.
- Native ETH will not be an asset supported by the strategy, as the current implementation does not support withdrawing it to the Vault.
- The strategy invests funds into Balancer MetaPools and Aura finance contracts, both of which are granted infinite allowance to spend the strategies' assets. Additionally, the `wstEth` and `sfrxEth` contracts are given `1e50` units of allowance to pull `stEth` and `frxEth` for wrapping purposes. The strategy contracts rely on all of these to function correctly and safely.
- The strategy will be set up so that only the strategist can trigger the deposit/withdrawal of funds.

***Update**: Because of a change during the fix review process, the calculation of BPT amounts during withdrawals/deposits now assumes that `frxETH` and `stETH` are pegged to `ETH`.*

Privileged Roles

- The governor can initialize the strategy.
- The vault can deposit and withdraw assets from the strategy.
- Either the vault, governor or strategist can change the `maxWithdrawalSlippage` and the `maxDepositSlippage`.

High Severity

H-01 `safeApproveAllTokens` Implementation Differs From Documentation

The `safeApproveAllTokens` function comments specify that the function should approve the Balancer pool to transfer all supported assets from the strategy, as well as approve the `wstEth` and `sfrxEth` contracts to pull `stEth` and `frxEth` from the Strategy for wrapping purposes. However, the function does not perform the necessary approvals to wrap the tokens, and additionally, it approves the Balancer vault and Aura rewards pool to withdraw an unlimited number of BPTs.

Consider correcting the `safeApproveAllTokens` function to follow its intended behavior.

Update: *Resolved in pull request #1776.*

H-02 `withdrawAll` Function Can Withdraw Unsupported Tokens

In the `BalancerMetaPoolStrategy`, the `withdrawAll` implementation retrieves the pool tokens and iterates through them, calculating the expected minimum amounts out. These calculated amounts are used for exiting the pool, where tokens are received in exchange for BPTs.

While these calculations are correct, if the intention is to withdraw across all tokens supported by the pool, they are incorrect when the assets supported by the strategy are a subset of the assets in the Balancer pool.

During initialization, the `BaseBalancerStrategy` passes a list of tokens to the `InitializableAbstractStrategy`, which in turn populates the `assetsMapped` array. This array contains the strategy's supported assets and is not guaranteed to include all assets supported by the associated Balancer pool.

When exiting, consider only withdrawing assets that are supported by the strategy to prevent ending up with assets that are not accounted for and could potentially become stuck.

Update: *Resolved. The Origin team stated:*

> *We decided not to make any changes. While we agree with the identified problem, our strategy currently does not support a configuration where only a subset of the pool's assets can be supported.*

H-03 `getBPTExpected` Calculation Is Not Robust

The `getBPTExpected` functions are used to calculate the minimum amount of BPTs to receive when depositing liquidity, or the maximum amount of BPTs to provide when withdrawing liquidity. They calculate the underlying ETH value of the assets to deposit/withdraw, and divide it by the price of a BPT in terms of the pool's underlying base asset.

There are two issues with this:

- Based on which oracles are being used, the strategy's asset price might be stale, which will result in an over/underestimation of the correct number of BPTs.
- The calculation assumes normal market conditions, namely that the pool's underlying base asset is trading at 1-1 parity with ETH. `stETH` or `frxETH` depegging from ETH would make this untrue, resulting in an overestimation of the amount of BPTs.

Overestimating the BPT amount can result in reverts during deposits while underestimating them can result in reverts during withdrawals. Consider accurately calculating the exact number of BPTs required for these operations using Balancer's StableMath.

Update: *Resolved in pull request #1795. The Origin team identified the additional concern of MEV attacks and stated:*

> *We have decided to move the primary responsibility of defending against MEV attacks back to the `ValueValueChecker` as our other strategies do. Only the strategist can deposit/withdraw funds and fork tests have been added confirming that the `VaultValue` checker used by the strategist catches MEV attack attempts. The checks in deposit/withdraw are now secondary sanity checks to prevent large losses and work without oracles. We can repay our debts in depegged coins, so we are not in danger of insolvency even if the pool mix changes. We did not implement the suggested math that would copy the math that the Balancer pool does under the hood since it would not protect the protocol against MEV sandwich attacks.*

Medium Severity

M-01 Non-Comprehensive Test Suite

The current test suite does not cover all the functionality in the `BalancerMetaPoolStrategy`.

- There are no tests for `deposit`, only for `depositAll`. This can be problematic, as supplying only one token at a time will incur more slippage and potentially return fewer BPTs than expected, as opposed to joining the Balancer pool with multiple supported assets at the same time.
- There are no tests for `safeApproveAllTokens`.
- There are no tests to ensure that `BAL` and `AURA` tokens are actually rewarded when calling `collectRewardTokens`.

Additionally, because of the significant number of external dependencies, there are multiple corner cases that are not covered in the existing test suite:

- If price oracles become stale and report outdated prices
- If `withdrawAll` is called when supported strategy assets are a subset of the pool's assets
- Based on different deposited/withdrawn values as well as oracle staleness, what `maxDepositSlippage` and `maxWithdrawalSlippage` should be in order to prevent DoS while depositing and withdrawing
- Whether the underlying value of the Strategy will be reported correctly irrespective of the liquidity pool being balanced

Additional tests would not only help maintain the correctness of the codebase when changes are implemented but would also serve as documentation for developers and auditors, aiding them in understanding multiple possible scenarios. Consider increasing the test coverage, especially around scenarios that deviate from normal operation.

Update: *Resolved in pull request #1780. The Origin team stated:*

> *1. The `deposit` test has not been added because single-asset deposits have been disabled.*
>
> *2. The test for `safeApproveAllTokens` has been added.*
>
> *3. The BAL and AURA harvesting test has been added.*

> *4. `maxDepositSlippage` and how it affects oracle prices: We are no longer using oracles so this portion is skipped. Changing `maxDepositSlippage` (now `maxDepositDeviation`) is tested in H-03. Also, the `maxDepositDeviation` is much more lenient now as `VaultValueChecker` has become the line of defense against MEV.*
>
> *5. Whether the underlying value of the Strategy will be reported correctly, irrespective of the liquidity pool being balanced: initial test added in H-03's pull request and enhanced via a separate commit.*
>
> *6. Stale oracle prices test: skipped since we do not use oracles.*
>
> *7. `withdrawAll` on a subset of assets: no test since this functionality is not yet supported.*

M-02 Withdrawal May Revert Because of Rounding Errors

The `_withdraw` function redeems BPTs from the Balancer pool in exchange for exact `_amounts` of tokens, which are then sent to the Vault. This is not a straightforward process, and involves at least two steps which introduce precision loss, potentially reverting the function upon `safeTransfer`.

1. Balancer has been found to round down by 1 WEI even when provided more than enough BPTs for the required liquidity. It has been observed to return only 1 less WEI so far, but that is not a guarantee. To circumvent this issue, the strategy overshoots the required liquidity by 2 WEI.
2. If the strategy asset to be withdrawn is a rebasing asset, it only exists as a wrapped version in the Balancer pool. Hence, the strategy calculates how much wrapped asset should be withdrawn from Balancer, withdraws it and then unwraps to the strategy asset. Because wrapping/unwrapping assets usually rounds in favor of the system (the `wstETH` and `sfrxETH` smart contracts), this method of calculation might also result in slightly fewer units of the required strategy asset.

Consider adding a fuzz test to have a greater certainty that rounding errors will not result in a denial-of-service of the `_withdraw` function, or requesting an error-adjusted amount of liquidity when exiting the Balancer pool.

Update: *Acknowledged, will resolve in pull request #1801. The Origin team stated:*

> *The pull request adds more fork test cases. We have fuzzing test infrastructure in development. Once that is done, we will add proper fuzzing tests, as suggested.*

M-03 Missing or Misleading Documentation

Throughout the codebase, there are multiple sections which are difficult to understand, and would benefit from additional or clearer documentation:

- It was difficult to assert that reward tokens are actually rewarded to the strategy. Consider adding more details around how this happens, especially for the AURA tokens where the process is more complex and not intuitive to follow.
- Given the intricacies of how Balancer works, as well as the need to wrap rebasing tokens, it was difficult to follow the math behind the `getBPTExpected` functions. Consider adding a practical example, as well as explanations regarding what each variable means and what each price is measured in.
- Consider documenting why the Balancer read-only reentrancy check is not needed on deposit and withdrawal functions.

Additionally, while there was documentation around the addition of strategies using Balancer, it was at times misleading as it was not specifically targeted towards the `BalancerMetaPoolStrategy`, and hence would differ from the implementation that was audited. Consider creating documentation specific to the strategy at hand, explicitly and concisely explaining the design decisions behind each important piece.

Update: *Resolved in pull request #1781 and pull request #1795.*

M-04 Providing or Withdrawing Liquidity One Asset at a Time Might Incur Considerable Price Impact

In the Balancer documentation, there is a section on risks which arise when using the platform. One section goes into detail about the potential loss of funds when entering and exiting pools.

```
##### Due to high price impact

When joining a pool, LPs should be aware of the price impact from adding tokens to the
pool. In general, adding liquidity in proportional amounts to the token weights of the
pool incur low price impact. Adding custom token amounts (non-proportionally) causes
the internal prices of the pool to change, as if you were swapping tokens. The higher
the price impact the more you'll spend in swap fees.
```

In the `BalancerMetaPoolStrategy`, there are multiple deposit function signatures, one of which deposits a single asset adding liquidity to the pool in a non-proportional amount. This function is integrated with the `VaultCore` via the `allocate` function.

Aside from the gas inefficiencies of sequential deposits, it is possible that, because of the swap fees incurred when swapping the provided token for all the others in the Balancer pool, considerably fewer BPTs will be returned than the allowed `maxDepositSlippage` and `maxWithdrawSlippage`, potentially DOSing single-token deposits and withdrawals. Hence, the `maxDepositSlippage` and `maxWithdrawSlippage` variables might need to be different, depending on whether the called functions deal with multiple pool assets at once.

Note that the above problem also applies when exiting a pool during withdrawals.

Consider always depositing/withdrawing all tokens simultaneously to avoid any reverts and inefficiencies as well as to comply with the best practices defined by the Balancer protocol.

Update: *Acknowledged, will resolve. The Origin team stated:*

> *We are aware of this shortcoming and have decided not to fix this issue. Thanks for being thorough and identifying it. Deposits are fine - we have only been planning on using `depositAll`, which does support multiple coins at once. We have disabled the single asset deposits for this strategy as they will never be used by `VaultCore`'s `allocate` function (we never plan on enabling this strategy as an asset default strategy). We plan on adding support for multi-asset withdrawals for all strategies in the future and Balancer's strategy will get the upgrade at that time, and with that, also gain the benefits of lower gas costs and better capital efficiency. `MaxWithdrawal` / `DepositSlippage` is now a lot more lenient (default setting of 1% instead of 0.1%). As per the solution in H-03, the `VaultValue` checker will be catching any irregularities (MEV attack attempts).*

Low Severity

L-01 Magic Numbers Are Used

In the `BalancerMetaPoolStrategy` contract, the value of `1e50` is used for setting approvals on both `stETH` and `frxETH`. There is no clear motivation as to why this number is selected, and the contract lacks inline documentation to clarify its significance.

Consider defining and documenting a constant variable to represent this number, as it is used in multiple places. Following Solidity's style guide, constants should be named in `UPPER_CASE_WITH_UNDERSCORES` format.

Update: *Resolved in pull request #1782.*

L-02 Naming Suggestions

Throughout the codebase, there are multiple variables named similarly but with vastly different meanings, which makes it difficult for the reader to understand the context. A non-comprehensive list of these is:

- When depositing or withdrawing, `_assets` and `_amounts` could be changed to `_strategyAssets` and `_strategyAssetsAmounts`.
- `mappedAmounts` and `mappedAssets` could be changed to `strategyAssetsToPoolAssets` and `strategyAssetsAmountsToPoolAssetsAmounts`. `mappedAssets` is especially confusing, as it is very similar to the strategy's `assetsMapped`, which has a completely different function.
- `asset` and `amount` could be changed to `strategyAsset` and `strategyAssetAmount`.
- `wrappedAssetAmounts` could be changed to `strategyAssetsAmountsToPoolAssetsAmounts`.
- `vaultAsset` could be changed to `strategyAsset`.
- `asset` could be changed to `strategyAsset`, or something consistent with the naming previously used.
- `assetAmount` could be changed to `poolAssetAmountToStrategyAssetAmount`.
- `poolAmountsOut` could be changed to `poolAssetsAmountsOut`.

In order to improve readability and better convey the code's intentions to the reader, consider modifying variable names to be more aligned with the codebase, while using these suggestions as potential guidelines.

Update: *Resolved in pull request #1783.*

L-03 Unused Function

The `checkBalance` function is only used in tests.

Consider removing it from the production code in order to improve readability.

Update: *Acknowledged, will resolve. The Origin team stated:*

> *We plan on using this read-only function in the future since it will be a much more gas-efficient way of calculating the vault's* `totalValue`*.*

L-04 Strategy Can Withdraw Unsupported Assets

In the `_deposit` function, tokens are checked to ensure they are supported before being deposited into Balancer pools. However, the `_withdraw` function lacks this check. If the assets supported in the strategy are a subset of the tokens in the Balancer pool, it can result in withdrawing an asset that is not supported by the strategy.

Consider only allowing callers to withdraw assets that are supported by the strategy.

Update: *Resolved in pull request #1784.*

Notes & Additional Information

N-01 Unused Files

The IERC20Details and IRETH files are unused.

Consider removing them to maintain a cleaner codebase.

Update: *Resolved in pull request #1785.*

N-02 Gas Inefficiencies

There are several places across the codebase where changes can be made to improve gas consumption.

- The `amount = 0;` assignment is unnecessary.

- There is another, more efficient version of the `fromPoolAsset` function that can be used when the amount is not needed. Consider switching to it instead.
- The `assetAmount` variable is unused, consider removing it.
- In the `withdraw` function, the conversion to a pool asset depends only on `poolAssets[i]`. Consider doing it only once per pool asset, outside of the `for` loop that goes through the strategy assets.
- The `asset` variable can be reused here, to avoid another array access through `_assets[i]`.
- There are several places where function parameters could be marked as `calldata`, consider changing them for future gas optimizations: `deposit`, `_deposit`, `withdraw`, `_withdraw`, and `getBPTExpected`.
- There are multiple instances of an array's length being accessed on every iteration of a loop. This can be made more efficient by saving off the length of the array in a local variable and using that in the `for` loop instead.

When performing these changes, aim to reach an optimal trade-off between gas optimization and readability. Having a codebase that is easy to understand reduces the chance of errors in the future and improves transparency for the community.

Update: *Resolved in pull request #1786.*

N-03 TODO Comments

The following instances of TODO comments were found in the codebase:

- The `TODO` comment on line 296 in BalancerMetaPoolStrategy.sol
- The `TODO` comment on line 380 in BalancerMetaPoolStrategy.sol

During development, having well-described TODO comments will facilitate the process of tracking and solving them. Without this information these comments may age and important information for the security of the system could be forgotten by the time it is released to production.

Consider removing all instances of TODO comments and instead tracking them in the issues backlog. Alternatively, consider linking each inline TODO to the corresponding issues backlog entry.

Update: *Resolved in pull request #1796.*

N-04 Inconsistent Function Naming

In the `BaseBalancerStrategy` some internal functions start with a leading underscore while others do not.

The use of standard conventions can quickly convey information to developers and users alike. Consider standardizing a naming convention for private and internal functions. A leading underscore is recommended, as mentioned in the Solidity documentation.

Update: *Partially resolved in pull request #1797. Some internal functions have not been updated with leading underscores ([1][2][3][4][5][6]).*

N-05 `mappedAssets` Is Initialized With Wrong Length

In the `_deposit` function, the length of `mappedAmounts` and `mappedAssets` is initialized to `tokens.length`. This is superfluous if only a subset of the Balancer pool's tokens are supported by the strategy.

While this currently does not present a security risk, consider initializing the array to the correct length.

Update: *Resolved in pull request #1800.*

N-06 Typographical Error

Consider addressing the following typographical error in the codebase:

- sent should be `send`.

Update: *Resolved in pull request #1798.*

Client-Reported

CR-01 Balancer Pool Tokens Can Remain in the Strategy After Withdrawing Funds

Withdrawing assets from the strategy calculates `maxBPTtoWithdraw`, namely how many BPTs should be redeemed with Balancer, in order to get the required liquidity back. Based on Oracle price staleness, this number might be slightly higher than the actual required amount. This means that there could be leftover BPTs in the Strategy contract, after most of them have been spent exiting the Balancer pool. This results in sub-optimal yield generation, as these BPTs should be re-deposited into Aura.

Update: *Resolved at commit b9dd480 and documented at commit 6ad405c.*

CR-02 `checkBalance` Assumes Normal Market Conditions

The `checkBalance` functions report the total amount of ETH inside the strategy, by multiplying the balance of BPTs held by their price, in terms of the pool's underlying base asset. The functions assume normal market conditions, meaning the underlying base asset is at 1-1 parity with ETH. This is not the case if `stETH` or `frxETH` were to depeg from `ETH`, in which case the `checkBalance` functions would report an inflated ETH value inside the strategy.

Update: *Partially resolved. Upon conducting additional research, the Origin team stated:*

> *The requirements we would like the `checkBalance` function to meet:*
>
> *- (critical) Flash loan pool manipulation must not change the total value of units (normalized assets) returned by looping through and adding all the supported pool asset balances together.*
>
> *- (important) `checkBalance` must report accurate amounts proportional to the share of assets held in the underlying Balancer pool.*

> *Conclusion: with the researched approaches, we are able to satisfy only one of the above requirements. We can borrow Balancer's pool math and calculate the precise amount of assets (the strategy is able to withdraw in any market condition). Unfortunately, such an approach is vulnerable to flash loan sandwich attacks that tilt the pool heavily toward one asset. And while the second requirement would be satisfied, the first one would not be.*
>
> *For this reason, we have decided to implement a solution that satisfies the first requirement and reports the same token balances no matter the share distribution of the assets in the underlying pool. This does not satisfy the second condition, which makes allocations and correctness regarding withdrawable underlying assets incorrect.*

Conclusion

Several high and medium-severity issues were identified. Additionally, suggestions have been made to increase the overall readability and cleanliness of the codebase.

We strongly advise implementing a comprehensive test suite around functionality calculating the underlying value of the strategy. Additionally, the provided documentation has been lacking and confusing at times, as it was not up-to-date with the implementation. We encourage employing better practices around documenting the design decisions behind key functionality and keeping such documentation up-to-date.