OpenZeppelin | security

OETH Integration

May 12, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

Security Model & Trust Assumptions 5

Medium Severity 7

M-01 Data feeds may be outdated 7

Low Severity 7

L-01 Missing docstrings 7
L-02 Unnecessary token allowance 8
L-03 Minting is still possible even if redemptions are not 8
L-04 Imbalanced token transfers 8
L-05 Inconsistent asset variables 8
L-06 Indefinite allowances 9
L-07 Possible rounding overcorrection 9
L-08 Potentially misleading platform token 9
L-09 Unsafe cast 10

Notes & Additional Information 10

N-01 State variable visibility not explicitly declared 10
N-02 Unused imports 10
N-03 Multiple contract declarations per file 11
N-04 TODO comments 11
N-05 Redundant use of the SafeMath library 11
N-06 Typographical errors 12
N-07 Incomplete generalization 12
N-08 require statements do not check for any conditions 12
N-09 Incorrect decimals type 12
N-10 Redundant code 13
N-11 Vault coupled to expected collateral 13

Conclusions 14

Appendix 15

Monitoring Recommendations 15

Summary

Type	DeFi
Timeline	From 2023-05-3 To 2023-05-11
Languages	Solidity

Total Issues	21 (0 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (0 resolved)
Low Severity Issues	9 (0 resolved)
Notes & Additional Information	11 (0 resolved)

Scope

We audited the following commits in the Origin-Dollar repository :

- The commit `508921bd39f988fa61b60cea372b910725ff7bd0` of pull request #1271.
- The commit `757ad13fa07ecdff058e55d9beff3c98d5d186a2` of pull request #1341.

In scope were the following contracts:

```
contracts
├── strategies
│   ├── ConvexEthMetaStrategy.sol
│   └── Generalized4626Strategy.sol
├── oracle
│   └── OracleRouter.sol
└── vault
    ├── VaultCore.sol
    └── VaultAdmin.sol
```

System Overview

The changes to the existing OUSD vault codebase introduce support for OETH, which is deployed independently. The code broadens the vault's functionality, enabling it to accommodate yield-generating collateral tokens instead of just stablecoins, so both vaults can use the same codebase. This is achieved by mapping collateral amounts to the corresponding "base token" (USD or ETH) before using them in calculations. The OETH vault will initially be collateralized with Rocket Pool ETH (rETH), Lido Staked ETH (stETH), Frax Ether (frxETH) and Wrapped Ether (WETH). These new tokens are priced using the corresponding Chainlink data feed. To learn more about how the vault works, refer to the following audit report.

We also reviewed a new `Generalized4626Strategy` contract, specifically engineered to support investing collateral into an ERC-4626-compliant vault. This will be used to invest the frxETH.

Lastly, we reviewed a strategy for investing WETH into a Curve pool, along with newly minted OETH, and then depositing the corresponding LP tokens into a Convex Gauge. This strategy is adapted from an existing Convex Finance strategy, explained in our previous audit report.

Security Model & Trust Assumptions

Minimal modifications were made to the Origin protocol governance system. Therefore, the assumptions are nearly identical to those listed in the published Origin Governance Audit and Origin Dollar Audit reports (1 and 2).

Slight deviations of note include:

- The system now staked ETH derivatives as collateral. Although the system has always relied on the collateral tokens maintaining their expected value, there is now an additional specific mechanism (i.e., slashing) that could undermine this assumption. The Origin team has reviewed the relevant tokens and concluded that this risk is minimal.
- The system relies on additional Chainlink price feeds to support the larger number of collateral tokens, which are presumed to function reliably and accurately.
- The new strategies invest funds in Curve's ETH-OETH pool and the Convex Finance Booster contract, and they are granted infinite allowance to spend the strategies' assets.

Therefore, the new strategies rely on these investment contracts to function correctly and safely.

- The new vault code removes the call to the automatic OGN Buyback mechanism. The `Buyback` contract is configured to only perform manual swaps anyway, so this should be functionally equivalent.
- When minting new OUSD or OETH, the price of the deposit (rather than the equivalent number of base units) determines whether the deposit is large enough to trigger a rebase or to allocate vault funds to strategies. This accounts for the possibility of unexpected collateral prices.

Medium Severity

M-01 Data feeds may be outdated

Chainlink's documentation recommends validating that prices returned by their feeds are recent and fall within reasonable bounds. The `price` function of the `OracleRouterBase` and the `OETHOracleRouter` contracts do not validate these properties directly, although the vault independently requires the price to be within 30% of the expected value. In addition, consider confirming the `updatedAt` parameter is suitably recent to ensure users can mint and redeem OUSD and OETH tokens at a reasonable market price.

Low Severity

L-01 Missing docstrings

Throughout the codebase there are several parts that do not have docstrings. For instance:

- Line 9 in `OracleRouter.sol`
- Line 57 in `OracleRouter.sol`
- Line 77 in `OracleRouter.sol`
- Line 125 in `OracleRouter.sol`
- Line 156 in `OracleRouter.sol`
- Line 159 in `OracleRouter.sol`
- Line 191 in `VaultAdmin.sol`
- Line 750 in `VaultCore.sol`

In addition, the `initialize` function of the `ConvexEthMetaStrategy` contract is missing the `@param` comment for its `_ptokens` parameter.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

L-02 Unnecessary token allowance

The ConvexEthMetaStrategy contract grants an allowance for the Curve pool to spend all WETH tokens. This allowance can also be renewed by the governor. However, the pool accepts ETH directly and never interacts with the WETH token.

In the interest of limiting the attack surface, consider removing this unnecessary allowance.

L-03 Minting is still possible even if redemptions are not

When redeeming OUSD or OETH, the corresponding amount of each collateral token is calculated. The operation will fail if any of the tokens drift too far from the expected price. However, when minting new OUSD or OETH, only the deposited collateral needs to be within the acceptable range. This introduces the possibility that users can deposit funds but will be unable to withdraw them.

In the interest of predictability, consider preventing deposits unless all collateral tokens are redeemable. This would help ensure that deposits and withdrawals are enabled and disabled together during unexpected market conditions.

L-04 Imbalanced token transfers

The `ConvexEthMetaStrategy` contract invests the exact deposit amount but withdraws *at least* the withdrawal amount. This discrepancy means there could be stray ETH remaining in the contract after each withdrawal. Although this ETH is accounted for in the contract balance, it won't be utilized until an administrator invokes the `withdrawAll` function.

Consider accounting for the contract balance when deciding how many LP tokens to redeem, so that any stray ETH is automatically reused.

L-05 Inconsistent asset variables

When initializing the `Generalized4626Strategy` contract, the generic asset variables are set and then the new token variables are redundantly set. If the contract is initialized with multiple asset pairs, or another asset/platform token pair is added, the existing `shareToken` and `assetToken` variables will be overwritten. On the other hand, if an asset is removed, the `shareToken` and `assetToken` will not be affected, and the contract will continue to allow

deposits and withdrawals using those tokens. This can lead to inconsistencies in both directions, where the contract incorrectly appears to support unsupported assets or vice versa.

A similar observation applies to the `ConvexEthMetaStrategy` contract, which has variables for WETH and the LP token, even though they are also recorded as the asset and platform token. In this case, the redundant variable will not be overwritten if a new pair of tokens is added, but the other inconsistencies still apply.

In the interest of consistency and predictability, consider using the existing variables to track the asset and share tokens, instead of introducing new ones. The `_abstractSetPToken` function can be used to ensure there is at most one active asset-share token pair.

L-06 Indefinite allowances

The `Generalized4626Strategy` and `ConvexEthMetaStrategy` contracts both approve token allowances for their respective platform contracts. They also include a mechanism for the governor to refresh those allowances. However, there is no mechanism to revoke the allowances if the platform becomes untrustworthy.

Consider allowing the governor to revoke all token allowances.

L-07 Possible rounding overcorrection

When withdrawing ETH, the `ConvexEthMetaStrategy` first calculates the number of LP tokens to burn. This calculation adds 1 to the conceptual calculation to account for possible rounding errors. However, this means that the calculated value could exceed the contract's LP token balance, and the contract would attempt to redeem more tokens than it has. Consider restricting the redeemed LP tokens to the contract's token balance.

L-08 Potentially misleading platform token

Investment strategies maintain a token mapping that tracks the platform-specific investment token corresponding to each invested asset. Typically, a user can review the deposit and withdrawal events to identify when funds are swapped for the platform token and vice versa.

However, the `ConvexEthMetaStrategy` invests funds in a two-step process, where ETH is deposited in a Curve pool and then the curve LP tokens are deposited to Convex finance. A user tracking token balances would consider the gauge tokens as representing the investment,

since the strategy does not hold any Curve LP tokens, but the events treat the Curve LP token as the platform token. Consider treating the gauge token as the platform token, or including another parameter in the events to distinguish the platform token from the strategy holdings.

L-09 Unsafe cast

The `OracleRouterBase` contract uses an unsafe cast to convert the signed price to a `uint256` type. Consider ensuring the price is positive before performing the type conversion. If desired, this could be achieved with OpenZeppelin's `SafeCast` library.

Notes & Additional Information

N-01 State variable visibility not explicitly declared

Throughout the codebase there are state variables that lack an explicitly declared visibility. For instance:

- `MIN_DRIFT` in `OracleRouter.sol`
- `MAX_DRIFT` in `OracleRouter.sol`
- `FIXED_PRICE` in `OracleRouter.sol`
- `shareToken` in `Generalized4626Strategy.sol`
- `assetToken` in `Generalized4626Strategy.sol`
- `MAX_INT` in `VaultCore.sol`

For clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

N-02 Unused imports

In `VaultCore.sol` the imports `Strings`, `IVault` and `IBasicToken` are unused.

In `convexEthMetaStrategy.sol` the import `Helpers` is unused.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

N-03 Multiple contract declarations per file

Within `OracleRouter.sol`, there is more than one contract, library or interface declared.

Consider separating the contracts into their own files to make the codebase easier to understand for developers and reviewers.

N-04 TODO comments

The following TODO comment was found in the codebase. This comment should be tracked in the project's issue backlog.

- The `TODO` comment on line 126 in VaultCore.sol

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

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

N-05 Redundant use of the `SafeMath` library

All solidity compiler versions higher than 0.8.0 implement native overflow protection, which makes using `SafeMath` operations unnecessary for regular arithmetic. Consider removing redundant usage of the `SafeMath` library from the `VaultCore` contract to improve code readability and save gas.

N-06 Typographical errors

The codebase contains some typographical errors:

- The `InitialiseConfig` struct should be `InitializeConfig` to be consistent with the `initialize` function that it is referencing.
- "seperate" should be "separate".

N-07 Incomplete generalization

The `Vault` code is intended to be generalized to support both OUSD and OETH use cases. However, there are still some values that are only relevant to one of the use cases. For instance:

- The token and several variables have "OUSD" in their name. This is just one example, but several comments, variables and events throughout the `VaultStorage`, `VaultAdmin` and `VaultCore` contracts reference "OUSD" or the `ConvexOUSDMetaStrategy`.
- Some of the comments (e.g., describing the value functions), reference "ETH".
- The redeem functions are described as returning stablecoins.

Consider generalizing the names and comments to apply them to both use cases.

N-08 `require` statements do not check for any conditions

Within `VaultCore.sol` there are multiple `require` statements that do not check for any conditions. For instance:

- The `require` statement on line 657
- The `require` statement on line 688

For clarity, it is recommended to use `revert()` in situations where `require()` does not perform any condition checks.

N-09 Incorrect decimals type

The new Vault functionality caches the token decimals for each supported asset. However, it unnecessarily casts to a `uint256` type from the `uint8` type specified by the ERC-20

standard. In the interest of consistency, consider retaining the `uint8` type. This will also ensure that `Asset` records only occupy a single storage slot.

N-10 Redundant code

There are several examples of redundant code. For example:

- The calculation that determines the amount of LP tokens to burn can be simplified to `(_wethAmount + 1) * k`. To see why there is no loss of precision, note that it only uses subtractions and multiplications, which always produce exact integer values if they do not overflow.
- This line retrieves the OETH balance unnecessarily.
- The `ConvexEthMetaStrategy` is initialized with an `_assets` array, but its only element (the WETH address) is already passed to the function.
- This conversion redundantly casts a `uint256` variable to a `uint256` type.

Consider simplifying or removing the redundant code.

N-11 Vault coupled to expected collateral

The `Vault` contract contains a mechanism to match collateral tokens to their equivalent "base" token amount. A similar conversion is used to standardize the price. However, although it is written generically, the `GETEXCHANGERATE` case is coupled to the rETH token. In particular:

- It assumes the existence of a `getExchangeRate` function, which is not part of a known standard. It would not support the cbETH token, for instance.
- It hardcodes an additional 1e18 conversion factor, which assumes the `getExchangeRate` function returns the ETH value of 1e18 collateral token units.

To simplify the vault, consider moving the token-specific logic into the `OracleRouter` contract or another adapter contract.

Conclusions

Several minor vulnerabilities have been found throughout the codebase, and fixes have been suggested. As indicated by the design issues described (with redundant variables or overfitted behaviors), the codebase could benefit from a methodical analysis of the existing system boundaries to determine the ideal way to extend and generalize it. The Origin team has been very responsive and willing to provide context and detailed explanations as needed.

Appendix

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, the Origin team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues affecting production environments. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring. In addition to the recommendations provided in the previous audit reports, it is recommended to monitor all sensitive actions that affect the new functionality, including:

- Monitor Chainlink oracles to detect unexpected changes between the price of collateral tokens and OETH.
- Monitor Rocket Pool and Lido (and other relevant staking providers) to detect potential slashing conditions that could diminish the value of the collateral.
- Monitor the health of the protocols the `Generalized4626Strategy` integrates with, including any unexpected changes to the supply or price of the underlying assets.
- Monitor the `mint`, `mintForStrategy`, `redeem` and `burnForStrategy` functions and any other function called directly or indirectly by a user, checking that the values returned or minted by 3rd party protocols are within certain well-defined boundaries (otherwise, consider them as suspicious transactions). This should include the amount of WETH-OETH LP tokens and corresponding Gauge tokens received, given the amount of collateral being deposited.