Origin Dollar Audit

OPENZEPPELIN SECURITY | NOVEMBER 4, 2022

Security Audits

November 1st, 2022

This security assessment was prepared by **OpenZeppelin**.

Table of Contents

Summary

The Origin Protocol team asked us to review and audit several additions to their Origin Dollar smart contracts. Below we present our results.

Type	DeFi – Rebasing stablecoin
Timeline	From 2022-10-03 To 2022-10-14
Languages	Solidity
Total Issues	14 (9 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	9 (5 resolved, 1 partially resolved)
Notes & Additional Information	4 (3 resolved)

Scope

We audited the OriginProtocol/origin-dollar repository at the `bfe0ac8e5d7c05b9bf1021fafb25e0aed8a6ed45` commit.

```
├── strategies
│   ├── BaseConvexMetaStrategy.sol
│   ├── BaseCurveStrategy.sol
│   ├── ConvexGeneralizedMetaStrategy.sol
│   └── ConvexOUSDMetaStrategy.sol
├── utils
│   └── StableMath.sol
└── vault
    └── VaultCore.sol
```

System overview

Origin Dollar (OUSD) is an ERC-20 compliant stablecoin backed by USDT, USDC, and DAI. Users can mint OUSD by depositing any of these assets to the system's vault, to later be invested in different strategies, such as AAVE and Compound lending pools, or the Curve 3Pool pool. The accrued interest, fees generated and reward tokens collected from these strategies are then distributed among OUSD holders through a rebasing mechanism, increasing their overall OUSD balance. To learn more about how the system works and how strategies are managed, refer to the previous audit report.

The audited commit introduces two new strategies for generating yield: The Convex Generalized Meta Strategy and the Convex OUSD Meta Strategy. Both take advantage of the Convex Finance platform to increase the yield on stablecoin deposits in the Curve system.

In particular, depositors to the Curve 3Pool receive 3CRV tokens to represent their share of the pool. Those tokens are then deposited to Curve's OUSD-3CRV metapool to potentially earn trading fees. Finally, the metapool LP tokens (OUSD3CRV-f) are deposited in Convex Finance to earn staking rewards. Naturally, the process is reversed when the strategies need to return the stablecoins to the vault.

The difference between the strategies is that OUSD Meta Strategy also mints new (uncirculated) OUSD to deposit into the OUSD-3CRV metapool to keep the metapool balanced (with some limits in extreme scenarios). Those funds are burned when they are withdrawn, so they should not be part of the circulating supply unless they are bought from the metapool directly. Additionally, OUSD

Security model and 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.

Slight deviations of note include:

- The rewards are now collected from the strategies by a "harvester" address, set by the governor on a per-strategy basis. As before, the governor can set the reward tokens arbitrarily which lets them withdraw any funds held by the strategies.
- The new strategies invest funds in Curve's OUSD-3Crv metapool 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.

Client-reported finding

While completing the audit fixes, the Origin team identified and alerted us to another issue.

The `BaseConvexMetaStrategy` contract initializes the maximum slippage when the variable is declared. This will set the variable in the implementation contract but not in the proxy contract where it is needed. The maximum slippage will default to zero, which will be excessively strict when redeeming metapool tokens and could cause the withdrawal to fail until the variable is updated.

This was fixed in commit `1c524c97ac4e70dbf09447313643a2a53e2da35d`.

Findings

Here we present our findings.

Medium Severity

balance without validating it matches the expected value. This has two implications:

- It does not account for tokens that could be sent to the contract directly.
- It assumes the contract has received the expected number of tokens.

The most important example is both instances when the `ConvexOUSDMetaStrategy` attempts to burn all its OUSD tokens. This would fail if its balance exceeds the acceptable range, which could occur if someone sends OUSD directly to the strategy.

Additionally, in the interest of predictability:

- When the amount of tokens to deposit is already known, that value can be used instead of the contract's balance. Alternatively, only the difference needs to be minted in the first place.
- When the amount of tokens to receive is already known, it can be used as the minimum withdrawal amount.

Lastly, in the interest of local reasoning and robustness, consider explicitly confirming that the Curve protocol returns the expected number of tokens, rather than assuming it respects the stated minimum thresholds.

Update: *Fixed in commit `290c68fd25a1f2967324948398227684ec834597`. The Curve system is still assumed to return the expected number of tokens.*

Low Severity

Complicated accounting

The `withdraw` function of the `BaseCurveStrategy` determines the number of 3CRV tokens to burn by using the price to slightly overestimate the desired value, determining the corresponding amount of stablecoin, and then scaling down the LP amount linearly to match the required stablecoin value. Instead, consider directly querying the amount of LP tokens to burn with the `calc_token_amount` function, and then adjusting for fees. If desired, the amount could be validated with the `calc_withdraw_one_coin` function.

> *calculated ends up being more code than the current method.*

Inconsistent reference to pToken

The `BaseCurveStrategy` has two mechanisms to refer to the 3CRV token: the generic mapping it inherits from `InitializableAbstractStrategy` and its own local variable. We believe the local variable was introduced as a simplification, because all three assets correspond to the same platform token, but it is used inconsistently. In particular:

- The `withdraw` function equivocates between the mapping and the local variable in the event emission and balance check.
- The `checkBalance` function uses the mapping to retrieve the platform token balance but then assumes the asset is worth one third of the value. This only makes sense if there are three assets mapping to the same platform token.
- There are code comments that explicitly account for the possibility that not all assets are mapped correctly.

Consider using the local variable throughout the contract and disabling the ability to set the mapping individually.

Update: *Partially fixed in commit `49ee19e1d6bbc623bb88027c997658d21fe32390`. Some functions (like `safeApproveAllTokens`) still use the mapping.*

Inconsistent rebase bound

When minting OUSD tokens, there will be a rebase if the amount equals the threshold. However, when redeeming or burning tokens, the amount needs to be strictly greater than the threshold. Consider using an exclusive (or inclusive) bound throughout the code base.

Update: *Fixed in commit `e574b38ba22ae95302b9386d06aed980337c07bc`.*

Incorrect event parameter

Both emissions of the `Deposit` event in the `BaseCurveStrategy` contract use the `platformAddress` (after redundantly casting it to an `address` type) as the `_pToken` parameter. Consider using the `pTokenAddress` variable instead.

The code base contains several misleading comments:

- The `collectRewardTokens` function of the `BaseConvexMetaStrategy` contract claims to send rewards to the vault, but it sends them to the `harvester` address.
- The `depositAll` function of the `BaseCurveStrategy` contract does not accurately describe how all descendant contracts retrieve the 3CRV token address.
- The comment explaining how much OUSD to add in the `ConvexOUSDMetaStrategy` states that it could mint less OUSD if the metapool has too much, but it always adds at least as much OUSD as 3CRV. Moreover, it claims the metapool will end up balanced. However it won't add more than twice as much OUSD (by value) than 3CRV, even if that's necessary to balance the metapool.
- The `mintForStrategy` function of the `VaultCore` contract claims that it cannot use the `nonReentrant` modifier because it clashes with the `BaseCurveStrategy` contract's modifier. However, the two modifiers do not interact with each other. Instead, it is the `nonReentrant` modifier on the `allocate` function that would cause the conflict.
- The `@param` comment describing the `_lpWithdraw` function of the `ConvexOUSDMetaStrategy` contract claims it is the number of Convex LP tokens to redeem, but it is the number of 3CRV tokens to retrieve.

Consider correcting these comments.

Update: *Fixed in commit `4b7f103658656a16d160231f96f36ac1d4336dfd` and commit `d7ddbb3a2c261ed600d99bc73377de53b4859ad5`.*

Missing docstrings

The `setMaxWithdrawalSlippage` function of the `BaseConvexMetaStrategy` contract is missing its `@param` statement. Consider including it.

Update: *Fixed in commit `689d527252ec78932d4bf422be3ac6b8245a777d`.*

functionality to grant addresses an infinite allowance. However, they bypass the safety mechanism in `safeApprove` that prevents changing the allowance between two non-zero values, which is intended to prevent front-running attacks that spend both allowances. In this case, since the intended allowance is unlimited, the possibility of front-running is irrelevant.

Nevertheless, we consider it bad practice to use a safety mechanism while bypassing its additional requirements. Consider using the standard `approve` function and validating that it succeeds.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *We need to handle USDT's non-standard return value (no boolean), and safeApprove provides a clean way of doing this.*

Inconsistent storage gaps

When using the proxy pattern for upgrades, it is common practice to include storage gaps on parent contracts to reserve space for potential future variables. The size is typically chosen so that all contracts have the same number of variables (usually 50). However, the code base uses inconsistent sizes and does not always include a gap at all. In particular:

- The `InitializableAbstractStrategy` contract reserves 98 slots, bringing the total storage usage up to 106.
- The `BaseCurveStrategy` has no storage gaps.
- The `BaseConvexMetaStrategy` contract reserves 30 slots, bringing the total storage usage up to 39.

Consider using consistently sized storage gaps in all contracts that have not yet been deployed. For contracts that cannot be changed, because they are ancestors of live contracts in the code base, consider documenting the unusual storage size to facilitate safe upgrades.

Update: *Fixed in commit `e30022cd9fb7e815cb16ffd09a21c276f3980250`.*

Uninitialized implementations

The proxy contracts that represent the strategies should be initialized before they are used. However, it is good practice to initialize the implementation contracts as well to reduce the attack

Update: *Acknowledged, not resolved. The Origin team stated:*

> *Our implementation contracts have the owner set in the constructor at creation and outsiders cannot initialize them. We will transfer ownership of implementation contracts to the governance system.*

Notes & Additional Information

Constants not declared explicitly

There are some occurrences of literal values with unexplained meaning in Origin's contracts. For example, line 135 in `BaseConvexMetaStrategy` and line 212 in `BaseCurveStrategy.sol`. Literal values in the code base without an explained meaning make the code harder to read, understand, and maintain. This makes the code harder to understand for developers, auditors, and external contributors alike.

Developers should define a constant variable for every magic value used (including booleans), giving it a clear and self-explanatory name. Additionally, for complex values, inline comments explaining how they were calculated or why they were chosen are highly recommended. Following Solidity's style guide, constants should be named in `UPPER_CASE_WITH_UNDERSCORES` format.

Update: *Fixed in commit `46cfa3a8e0c9511b987234978178d80248215664`.*

Implicit visibility

The state variables `crvCoinIndex` and `mainCoinIndex` in the `BaseConvexMetaStrategy` contract do not have an explicit visibility modifier. In the interest of clarity, consider including it.

Update: *Fixed in commit `9d794d1e682dad35d61dd84b906b6168179a46ec`.*

Unindexed events

Update: *Acknowledged, not resolved The Origin team stated:*

> *We're going to keep our existing event signatures for backwards compatibility. For the new event (OusdMetaStrategyUpdated), we'll keep it consistent with the existing events.*

Unused import

The `BaseCurveStrategy` has an unnecessary import statement. Consider removing it.

Update: *Fixed in commit `ea5154feaa97b64a36bd49d83d38be6b9d526524`.*

Conclusions

No critical or high severity issues were found in the code base. Several minor vulnerabilities have been found and recommendations and fixes have been suggested.

Appendix

Monitoring Recommendation

Due to the high number of interactions between the system and third-party protocols through strategies such as AAVE, Curve and Compound, as well as the strong trust assumption towards the contract owner, the strategist and harvester roles, we recommend implementing monitoring for all sensitive actions, including but not limited to:

- Monitor the `mint`, `mintForStrategy`, `redeem`, `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 amount of cTokens, aTokens, 3CRV LP tokens and OUSD-3CRV LP tokens, given the amount of collateral being deposited.
- Monitor the health of 3rd party protocols to identify unusual situations that may put protocol funds at risk, including unusually large or frequent liquidations or unbalanced pools.
- Monitor all the functions that implement `onlyGovernor`, `onlyHarverster`, `onlyVaultOrGovernorOrStrat`

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin

Threat Monitoring
Incident Response
Operation and Automation

Zero Knowledge Proof Practice

Blog

Company

About us
Jobs
Blog

Contracts Library

Docs