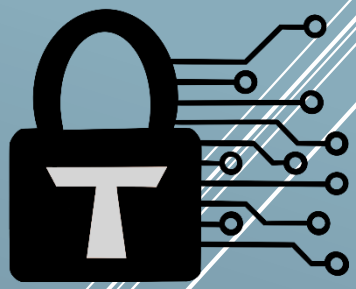
Trust Security



Smart Contract Audit

StakeDAO Curve Oracles

27/08/25

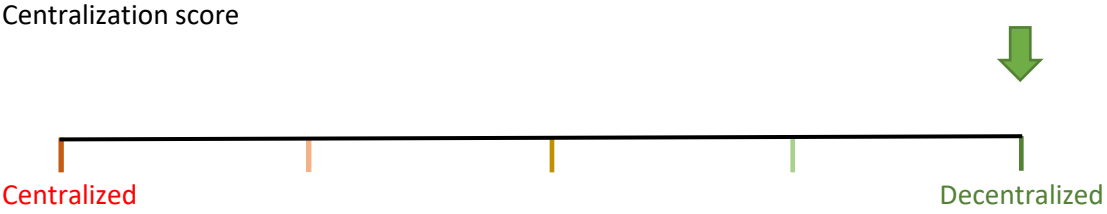
Executive summary

Category	Oracle
Audited file count	3
Lines of Code	174
Auditor	Trust SpicyMeatball
Time period	20/08/25 - 23/08/25

Findings

Severity	Total	Fixed	Acknowledged
High	1	1	-
Medium	2	2	-
Low	1	-	1

Centralization score



Signature

Digitally signed by Trust
Date: 2025.08.27 17:59:34 +03'00'

Document properties

Versioning

Version	Date	Description
0.1	23/08/2025	Client report
0.2	27/08/2025	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- BaseOracle.sol
- CurveCryptoswapOracle.sol
- CurveStableswapOracle.sol

Repository details

- **Repository URL:** https://github.com/stake-dao/contracts-monorepo/
- **Commit hash:** 9a4c9de69bee6b7286696bc8c1a56182a06d565f
- **Mitigation review commit hash:** 2f11d737bfa9dc705207aa33e9cffe3e699eaeb6

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

About the Auditors

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys sharing knowledge and experience with aspiring auditors through X or the Trust Security blog.

SpicyMeatball is a top leaderboard auditor, member of Code4rena Zenith, and has completed over 50 private audits.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project kept code as simple as possible, despite implementing custom data structures for efficiency. External integration with Curve introduces some complexity due to variety of pools introduced over time.
Documentation	**Excellent**	Project is very well documented including example configurations.
Best practices	**Excellent**	Project consistently adheres to industry standards.
Centralization risks	**Excellent**	Project does not introduce any admin permission structures.

Findings

High severity findings

TRST-H-1 The oracle overvalues pools that incorporate custom rates

- **Category:** Accounting issues
- **Source:** CurveStableswapOracle.sol
- **Status:** Fixed

Description

Stableswap NG pools introduced direct support for appreciating tokens (wstETH, sDAI etc), by applying the invariant after multiplying each asset by the rate, which is in **stored_rate**. In *get_virtual_price()*, the price is denominated in underlying, which would be USD for USD-stables, ETH for ETH-stakers, etc. If **coin0** is appreciating, the rise will overall be accounted for **twice**, once in the direct underlying rise above, and again when pricing **coin0**. This causes the LP value to appear larger and will misprice collateral.

Below is an example calculation from block 23203598, pool sUSDS-USDT .

Total Supply: 50251772351572841470481893
get_virtual_price(): 1013636809764308958
stored_rates: 1064360029237351595, 1000000000000000000000000000000
coin0: sUSDS
Savings USD val: 1.064
Total pool value by Etherscan: **$50,921,896**

Calculating total LP value in underlying:

(get_virtual_price() / 1e18) * (total_supply / 1e18) = 50,937,046
Converting to USD by plugging in sUSDS value:
50,937,046 * 1.064 = **$54,197,016**
Converting to USD by first dividing by the stored rate:
50,937,046 / 1064360029237351595 * 1e18 * 1.064 = **$50,919,816**

As can be seen, dividing by the stored rate results in a value just slightly under the Etherscan-computed value, whereas otherwise the result is massively overvaluing the LP pool.

Note that the **stored_rate** value is sensitive to decimals. It is comprised of a constant multiplier and a rate deduced from external logic per protocol. A summary of its properties is found below.

```
stored_rate[i] = rate_multiplier[i] * raw_rate[i] / 1e18

rate_multiplier[i] = 10^(36-decimals[i])

raw_rate[i] always in 18-decimals
```

Therefore, the overall behavior is that if asset is in **18-X** decimals, **stored_rate** is in **18+X** decimals. That value should be normalized when making the final calculation.

Recommended mitigation

In case the pool supports a stored rate, divide by it as demonstrated above, before returning the value in *_getLpPriceInCoin0()*.

Team response

Fixed.

Mitigation review

The issue correctly fixes the root cause of the problem. It also accounts for ABI mismatch with some unusual Curve pools.

Medium severity findings

TRST-M-1 Read-only reentrancy in get_virtual_price() could lead to mispriced assets

- **Category:** Reentrancy attacks
- **Source:** CurveStableswapOracle.sol
- **Status:** Fixed

Description

In 2022, a read-only reentrancy vulnerability was discovered in the original Stableswap pool implementation. During *remove_liquidity()* calls, the caller receives a callback in mid-state, making calls to *get_virtual_price()* incorrectly account for one of the assets. The issue has been addressed in modern pools by adding a reentrancy lock on the function, while integrators operating with older pools mostly handle them by implementing an external reentrancy check – calling a non-reentrant function like *withdraw_admin_fees()* before checking the price.

The provided Stableswap oracle does not have any protections when using the affected function. There is some indirect protection by calling *price_oracle()* in the pricing flow, a newer function which many older pools do not contain. However, some pools are not covered by the *price_oracle()* call, although none have been directly identified as containing ETH/ERC-777, a pre-requisite for the attack. However, a comprehensive check has not been made during the time-box of the audit, so it is unknown whether fully vulnerable pools exist.

Recommended mitigation

Add an external reentrancy check as mentioned above.

Team response

"Acknowledged. While this was an issue in the past, it appears that all modern pools compatible with our implementation (those that implement the required price_oracle method) are protected against this attack. The get_virtual_price function is, by default, safeguarded against reentrancy. Additionally, the Curve team has confirmed that this protection will remain for all future pools. We also could not find any modern pool in production that could be attacked as described.

However, the threat is serious, which is why it is crucial never to deploy an oracle if get_virtual_price is not protected. This recommendation has been added to the contract documentation in commit c189711. It is also included in our pre-deployment checklist and will immediately disqualify any misconfigured pools."

Mitigation review

Considered fixed as a responsible admin which will be acting on the latest documentation and knowledge base is assumed.

TRST-M-2 Oracle pricing calls could revert due to overflow in conversion loop

- **Category:** Overflow issues
- **Source:** BaseOracle.sol
- **Status:** Fixed

Description

The BaseOracle supports an arbitrary-length price feed array. When pricing, the decimal units of the feeds accumulate on the numerator of the calculation, and at the end are neutralized using the **SCALE_FACTOR**.

```
for (uint256 i; i < length; i++) {
    uint256 feedPrice = _fetchFeedPrice(token0ToUsdFeeds[i],
token0ToUsdHeartbeats[i]);
    numerator = numerator * feedPrice;
}
```

The issue is that during the intermediate stage, **numerator** could overflow from combining the LP value in coin0 units, with the accumulated oracle feed decimals. For example, if **A = 200,000 B**, while pricing A in B through four 18-decimal oracles, the result would be **2e(5 + 4*18)**, overflowing uint256. Note that in the worst case, the overflow would not be detected during testing of the oracle, and only when the value of A rises on-chain, would the overflow be triggered.

Recommended mitigation

Consider dividing by the feed decimals at every loop iteration. This introduces a small divide-before-multiply issue, but the effect is very minor and rounds conservatively, in the desired direction.

Team response

Fixed.

Mitigation review

Issue has been addressed as recommended.

Low severity findings

TRST-L-1 The StableSwap oracle is overly conservative and reduces capital efficiency

- **Category:** Logical flaws
- **Source:** CurveStableswapOracle.sol
- **Status:** Acknowledged

Description

In *_getLpPriceInCoin0()*, the StableSwap oracle calculates the value in underlying units by *get_virtual_price()*, and multiplies by the lowest value between the member coins. It is essentially assuming the entire value is in the weakest token in the pool, which is unrealistic. For example in the case of an 8 stablecoin pool, 7 of 8 of the tokens would be undervalued by an arbitrary amount, causing the overall price of the LP to be pessimistic and limiting the capital efficiency of the overall solution.

Furthermore, when the loan token is **coin0**, the value returned by *get_virtual_price()* is by definition the amount of **coin0** LP is quoted at, so converting via the weakest token is incorrect.

It is understood that using the minimum value is a method that has been used by some protocols, and provides a strong guarantee that the oracle does not ever overvalue assets, but if capital efficiency and accuracy of measurement is of importance, a more fine-grained approach is recommended.

Recommended mitigation

Multiply the *price_oracle()* reading of each token with the relative part of the pool held in it. If **coin0** is the loan asset, do not perform any minimization.

Team response

Acknowledged. We'll stick to our conservative approach. At that point, the strong guarantees implied by our current approach are desired.

Additional recommendations

TRST-R-1 Avoid integration with pools that have pricing issue

The Curve documentation warns of a bug in Stableswap-NG that can cause mispricing of assets when the tokens have different decimal precisions, or use external rates. Ensure that those pools are never used.

Systemic risks

TRST-SR-1 Curve Pools Integration

The integration with Curve Pools introduces several risks:

- In Curve there is a significant variety of pools, and even the documentation does not mention all the different pool implementations over the years. Pools could have surprising behaviors, like internalizing public variables, having a mix of various pool types, and many others. This reinforces the importance of adequate testing for each supported pool.
- General risks of external bugs and all known and unknown price manipulation attacks are carried over.

TRST-SR-2 Chainlink Integration

The Chainlink integration is trusted, and immutable on the StakeDAO oracle side, so any issues would surface upwards.