PUBLIC

Code Assessment

# of the Curve & Convex Feature Smart Contracts

April 4, 2023

Produced for

Silo

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Silo team,

Thank you for trusting us to help Silo Finance with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Curve & Convex Feature according to Scope to support you in forming an opinion on their security risks.

Silo Finance implements a Curve LP price feed for StableSwap and Crypto pools, a price feed using Curve's pool prices, and a forwarding price feed that maps assets to another asset. Further, Silo Finance provides an ERC-20 wrapper for Convex staking positions. Last, Silo Finance created a new implementation for the Silo router so that it supports wrapping and unwrapping the wrapped tokens.

The most critical subjects covered in our audit are asset solvency, access control, functional correctness and oracle robustness. Security regarding all is high.

The general subjects covered are gas efficiency, documentation, unit testing and trustworthiness. Security regarding all is high. However, some can be improved (e.g. gas efficiency).

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	2
• Code Corrected	2
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	7
• Code Corrected	6
• Acknowledged	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Curve & Convex Feature repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	9 Jan 2023	3d5563302739b3ab953fd037373e3b9777e100fd	Initial Version
2	6 Mar 2023	282fbecb20446dea34ef5d385b5b15f10569fd7c	After Intermediate Report
3	18 Mar 2023	05571278056516934f66d710aed33e0cba345195	Third iteration
4	29 March 2023	b7cfb81b867f399f5ac94a59387669cd005e1a6e	Read-only reentrancy and Convex changes
5	4 April 2023	fd70c308e73692abd756a0c4e9a8f2fed3669adf	Removal of unused error

The smart contracts in scope are:

```
contracts/priceProviders/curve/*
contracts/priceProviders/curveLPTokens/*
contracts/external/convex/*
contracts/wrappers/*
contracts/priceProviders/forwarder/ForwarderPriceProvider.sol
contracts/SiloRouterV2.sol
```

For the solidity smart contracts, the compiler version `0.8.13` was chosen for all contracts besides the contracts for the Convex wrapper for which `0.6.12` was used.

In (Version 2), the following files have been removed and are out of scope:

```
contracts/priceProviders/curve/*
```

In (Version 2), the following file renamings occured:

```
contracts/wrappers/convex/ConvexStakingWrapperSilo.sol -> contracts/wrappers/convex/ConvexSiloWrapper.sol
contracts/wrappers/convex/ConvexStakingWrapperSiloFactory.sol -> contracts/wrappers/convex/ConvexSiloWrapperFactory.sol
```

In (Version 4), the following files were added:

```
contracts/priceProviders/curveLPTokens/peggedAssetsPools/CurveReentrancyCheck.sol
contracts/priceProviders/curveLPTokens/interfaces/ICurveReentrancyCheck.sol
contracts/priceProviders/curveLPTokens/interfaces/ICurveHackyPool.sol
```

2.1.1 Excluded from scope

All other contracts in the system are out of scope. Convex is out of scope. Curve is out of scope. The parameter selection is out of scope. Note that we expect that the assets using the price-providers can only be used as collateral.

In (Version 2), some smart contracts that contained logic to fix some reported issues of (Version 1) were added. Note that only the parts associated with the fixes were reviewed and the new smart contracts were not audited.

In (Version 3), Silo Finance specified that a change of the collateral vault in the convex wrapper may lead to a loss of rewards and is intended.

In (Version 4), Silo Finance added a read-only reentrancy detection mechanism which was included in scope. The selection of parameters for this is out-of-scope.

2.2 System Overview

This system overview describes the initially received version ((Version 1)) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Silo Finance offers oracle implementations for the Curve protocol and an ERC20 wrapper for Convex staking positions to introduce ERC-20 capabilities so that these become usable in the context of Silo while still receiving Convex rewards. Additionally, forwarding price feed is introduced and a new router version is provided implementing one-click actions for Convex.

2.2.1 Price Providers

Silo Finance introduces several new price providers that implement the `IPriceProvider` interface with the `getPrice()` function that returns the price of an asset denominated in the quote token of the `PriceProviderRepository` smart contract. This function typically first computes the price in a local quote asset and then converts the quote main quote token of the price provider repository using the repository's `getPrice()` function.

The `CurvePriceProvider` and the `CurvePriceProviderETH` provide spot prices for specific assets using Curve's `get_dy()` function. Those assets need to be configured by the manager, a privileged address, beforehand with the `setupAsset()` method to set up the local quote asset for the queried asset.

Further, price providers for Curve LP tokens are implemented. Namely,

1. for StableSwap pools (pegged assets)
2. and for CryptoSwap pools (non-pegged assets)

The LP price estimation for StableSwap LPs performs a lower-bound estimation by multiplying the virtual price (`get_virtual_price()`) with the minimum price of the underlying assets. Hence, the local quote asset will be the underlying asset with the lowest value. Note that `setupAsset()` whitelists LP tokens and ensures that an LP token's underlying tokens are supported by the protocol. Note that LP tokens may have other StableSwap LP tokens as underlying (meta pools).

The LP price estimation for CryptoSwap pools is computed by using the `lp_price()` function. Hence, the local quote asset is the asset at index 0. Note that one other particular case is handled for Ethereum mainnet: the tricrypto2 pool, which has an `lp_price()` function implemented in a separate smart contract.

Note that setting up LP tokens in the pegged or non-pegged Curve LP price providers requires some details of the LP tokens. These are queried through so-called fetcher contracts where the main entry point is the fetchers' repository `getLPTokenDetails()` function that forwards this request to the individual fetchers. These individual fetchers query the official curve registries and factories to find information about the pools. More specifically, fetchers for the main registry, the metapool factory, the crypto registry, and the crypto factory are implemented.

Last, a forwarding price feed is implemented that maps an asset to another asset so that a 1:1 rate between the two is enforced.

2.2.2 Convex Staking Wrapper

The `ConvexStakingWrapper` is a smart contract that wraps a Convex staking position into an ERC20 token. Note that the convex staking position's underlying is a Convex token which itself represents a Curve gauge staking position.

For users to receive the wrapper tokens, they can either `stake()` (stake Convex tokens) or `deposit()` (deposit Curve LP tokens and stake them). To withdraw, users can either receive the Curve LP token directly by calling `withdrawAndUnwrap()` or the convex token with the `withdraw()` function.

Rewards can be withdrawn using `getReward()` by anyone for any address. Note that Convex supports multiple reward tokens. Consequently, the wrapper takes those into account to distribute rewards fairly among token holders. To achieve this, it keeps track of the latest reward per token for each reward. Per user, the reward per token is checkpointed on every balance change. Note that checkpointing is in all balance-changing functions and can also be done using the external checkpointing functions.

The `ConvexStakingWrapperSilo` is a modified version of the wrapper that allows users to claim their Convex rewards while using the token as collateral in the Silo finance protocol. It queries the collateral balance of the corresponding silo for the user and adds it to the user's current balance to get the total balance to distribute rewards proportionally.

Silo Finance also provides a `ConvexStakingWrapperSiloFactory` that allows anyone to create a `ConvexStakingWrapperSilo` for a specific Convex pool id. Note that there is only one wrapper per pool id.

2.2.3 Router V2

The `SiloRouterV2` is the second version of the `SiloRouter`. Silo routers are helper smart contracts that are used to batch actions to facilitate the user experience. They allow multiple actions to be executed in only one transaction and are particularly useful for the newly introduced Curve LP staking wrapper.

In router V2, Silo Finance added external actions so that users can batch wrapping/unwrapping actions with some other Silo protocol-specific actions.

With the current state, the only possible external actions are to wrap and unwrap between Curve LP tokens and the corresponding protocol's `ConvexStakingWrapperSilo` tokens.

2.2.4 Roles & Trust Model

Manager: Fully trusted. Has access to privileged functions on price feeds. Could set up bad price feeds. We assume that only legitimate and suitable pools will be set up.

Owners: Fully trusted. Can shut down a wrapper contract.

Silo system: Fully trusted and expected to work correctly.

Curve pool: Fully trusted to work correctly. This price feed only works for Curve Pools versions 2 and 3 (StableSwap and CryptoSwap). In particular, version 1 pools are not supported by this price feed. However, exceptions in interfaces could occur which could be unsupported. Furthermore, the use of the virtual price implies the assumption that the pools are balanced, which they will not be at all points in time. In case some Curve-Lending Pools are used, the protocol receiving the actual funds is also trusted to work correctly. Incorrectly set up pools, e.g. certain factory-produced pools, will not work correctly. Further, we expect the LP price returned by Curve's CryptoSwaps to be in 18 decimals notation.

Convex system: Fully trusted to work correctly. Assumed to not suddenly start reverting. Assumed to have no rebasing and double entry point tokens as well as tokens with fees as reward tokens. Further, it is assumed that reward tokens among reward pools are distinct. Also, we expect the number of reward tokens to be very small.

Users: Untrusted.

2.2.5 Changes In Version 2

- The Curve spot price price providers have been removed.
- The shutdown logic has been changed. If the shutdown is activated, checkpointing and calling the extra rewards hook is possible now while interacting with Convex is skipped.
- A new assumption is introduced: If the total amount of underlying tokens of a Metapool LP exceeds eight (including nested ones), the code will revert which is expected.

2.2.6 Changes in Version 3

It has been specified that when the collateral vault variable is updated through `syncSilo()`, rewards are still lost. Given the low severity of this issue, we resolved it.

2.2.7 Changes in Version 4

Silo Finance added a read-only reentrancy protection mechanism to the pegged asset price provider. Namely, the protection mechanism works by calling the pool's `remove_liquidity()` function and measuring its gas consumption. If the gas used is below a certain threshold, it is assumed that the pool's reentrancy lock has been triggered. Otherwise, it is assumed that the the reentrancy lock has not reverted. Note that governance must set up the parameters when a new Silo is supported. Note that we exclude this setup along with potential hard fork changes from scope. Further, note the potential corner cases of the mechanism, see Read-only reentrancy protection.

Further, the Convex wrapper has been adapted to handle the wrapper contracts for extra rewards.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Some CryptoSwap Pools Do Not Implement the lp_price() Function Acknowledged

5.1 Some CryptoSwap Pools Do Not Implement the `lp_price()` Function

Correctness Low Version 1 Acknowledged

Not every CryptoSwap pool of the Curve protocol implements the `lp_price()` function that is used in the `CurveNPAPTokensPriceProvider`. The only exception that was added is the tricrypto2.

Acknowledged:

Silo Finance replied:

```
We will implement them if we need it. For now we need only tricrypto2.
```

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical -Severity Findings	2

- Curve LP Oracle Is Vulnerable to Read-Only Reentrancy Attacks Code Corrected
- CurvePriveProvider Uses the Spot Price Code Corrected

High -Severity Findings	1

- Collateral Token Transfers Are Not Taken Into Account Code Corrected

Medium -Severity Findings	1

- Missing Shutdown Logic Code Corrected

Low -Severity Findings	6

- A Metapool Could Have More Than One Nested LP Token Code Corrected
- Metapool Setup Recursion Lacks Sanity Check Code Corrected
- Metapools With Two LP Underlying Code Corrected
- Missing Events for State Modifying Actions Code Corrected
- The Reward Integral Computation Can Overflow Code Corrected
- IWrapperDepositor Not Implemented Code Corrected

6.1 Curve LP Oracle Is Vulnerable to Read-Only Reentrancy Attacks

Design Critical Version 1 Code Corrected

The Curve LP oracle smart contracts can be manipulated by using the read-only reentrancy vulnerability. This is because no checks are done regarding the reentrant state of the curve pool. For further details please see our blog post: https://chainsecurity.com/curve-lp-oracle-manipulation-post-mortem/.

Code corrected:

A protection mechanism has been implemented.

6.2 CurvePriveProvider Uses the Spot Price

Security Critical Version 1 Code Corrected

The `CurvePriceProvider` smart contract uses the spot price of the curve pool to get the price of assets. This is done by calling the `get_dy()` function on the curve pool.

Hence, an attacker could easily manipulate the price with a flash loan or a lot of liquidity. If this oracle is used as a source of truth for a borrowing or liquidation mechanism, funds could be stolen from the protocol.

Further note, that also the `IPriceProvider`'s NatSpec is not accurate in that case as it specifies that the TWAP is calculated.

Code removed:

All related code has been removed. Curve is not used as a price provider anymore.

6.3 Collateral Token Transfers Are Not Taken Into Account

Correctness High Version 1 Code Corrected

The balance used for payouts is the sum of the ERC-20 balance and the user's Silo collateral-only shares (converted to the ERC-20). Hence, each change in one of these balances must ensure the correctness of payouts. While the ConvexStakingWrapperSilo contract performs reward checkpointing for each ERC-20 balance change, the integration with the Silo fails to integrate such checkpoint fully, thus allowing both theft and loss of rewards.

A lack of checkpointing is present

- When a user transfers the collateral tokens: Transferring collateral tokens does not trigger any checkpointing.
- When a user uses the `depositFor()` function in the silo: The checkpointing will be done for the transfer, however, minting collateral tokens will not checkpoint for the recipient.
- When a user is being liquidated: The liquidated user will not be involved in checkpointing and loses his rewards. The liquidator does not receive the rewards as a liquidation bonus.
- When a user uses the router to withdraw the collateral: The router will be checkpointed. The user will temporarily get a smaller balance accounted which will lead to unfair checkpointing for the user.

Most notably, the collateral token transfers would allow for a simple attack by transferring collateral tokens from address to address to claim rewards with each.

Further, note that the function `syncSilo()` could change the silo which is used for additional accounting. However, that could break the contract and unfairly distribute rewards.

To summarize, collateral token transfers do not checkpoint and transfers of tokens between the silo and a user, who was not the prior owner of these tokens, update the rewards only for the receiver.

Code corrected:

Silo Finance has added a Silo type called `SiloConvex` which updates reward for concerned users before any actions. Router, current silo and deprecated silos cannot be checkpointed for rewards anymore.

Further, `ShareCollateralTokenConvex` has been introduced that checkpoints before any direct transfer between users.

Note that when the `collateralVault` variable is updated through `syncSilo()`, some rewards could still be lost, but this is now part of the specification.

6.4 Missing Shutdown Logic

Design Medium Version 1 Code Corrected

Each ConvexStakingWrapper smart contract can be shut down by the owner so that accounting of the rewards stops and users are only able to withdraw their shares of the pool. Rewards will not be accounted for anymore when moving wrapped tokens around because the _checkpoint() function does not apply any logic when the isShutdown flag is set to true.

However, the _checkpointAndClaim() function does not have any such check for the flag, meaning that users are still able to claim their rewards after the shutdown. Users could also still transfer the wrapped tokens without the wrapper checkpointing it so an attacker could manipulate its balance to steal some rewards.

Code corrected:

Both functions now implement the shutdown logic.

6.5 A Metapool Could Have More Than One Nested LP Token

Correctness Low Version 1 Code Corrected

The CurvePAPTokensPriceProvider retrieves all the underlying tokens for the LP token. For meta pools, the underlying LP token's coins are retrieved.

However, when the nested depth is equal to or greater than 2, it directly tries to fetch the price of the last lp token instead of continuing to fetch the underlying tokens.

Code corrected:

Code now correctly fetches all nested tokens recursively. Note that if the total amount of underlying tokens exceeds eight, the code will revert which is expected.

6.6 Metapool Setup Recursion Lacks Sanity Check

Correctness Low Version 1 Code Corrected

Metapools are StableSwap pools where at least one underlying is a Curve LP token. To set up such a metapool LP token, setupAsset() in CurvePAPTokensPriceProvider recursively sets up all underlying LP tokens. However, the recursive iterations of _setUp() lack the _MIN_COINS sanity check and the LPTokenEnabled event emission.

Code Corrected:

The check and the event emission are now in the _setUp() function which is used for recursively iterating over the potentially nested LP tokens.

6.7 Metapools With Two LP Underlying

Design Low Version 1 Code Corrected

Note that Metapools can have only one LP underlying. However, the current design would allow for bad pools with two LP tokens as underlyings - one as a regular coin and one as the base asset.

Code corrected:

The code now reverts if there are two lp underlyings.

6.8 Missing Events for State Modifying Actions

Design Low Version 1 Code Corrected

In `ConvexStakingWrapper`, some important state-modifying actions do not trigger events:

- Shutting down a staking wrapper
- Adding a reward token
- Setting the hook
- Checkpointing users

Emitting events could ease following the state of the contract.

Code corrected:

Events have been added for all of the above.

6.9 The Reward Integral Computation Can Overflow

Correctness Low Version 1 Code Corrected

`ConvexStakingWrapper` uses solidity `0.6.12` which can overflow on arithmetic operations. The staking wrapper uses a variable called `reward_integral` to track each token reward by increasing it proportionally to the received rewards. Note that this variable should only be able to increase.

At line 275, it computes it in such a way:

```
reward.reward_integral = reward.reward_integral + uint128(bal.sub(reward.reward_remaining).mul(1e20).div(_supply));
```

The addition here can overflow in some conditions, which would lead to a decrease of `reward_integral`. A decrease in the variable would mean that some users would have their rewards locked forever in the smart contract.

Also note that the truncation to a `uint128` in the second part of the code line can also lead to a truncation overflow, which would miscompute the received rewards.

While the supply is low, the overflow can be triggered very easily. Using this capability, an attacker could grieve users from receiving rewards by pushing the users' reward integrals to the maximum so that the global reward integral cannot exceed the users' reward integral.

Code corrected:

The reward variables are now stored as `uint256`.

Silo Finance noted:

```
Contract is updated to store the rewards
integral in 256 bits variables. This fix makes the integral overflow
improbable for regular reward tokens, even if the total supply of the
wrapped token is 1 wei.
```

Here `reward_integral` could still overflow but as stated by Silo Finance, it is much less probable given that reward tokens should not be of very low value or with unusually high decimals.

6.10 `IWrapperDepositor` Not Implemented

Design Low Version 1 Code Corrected

Most of the contracts interact with each other based on the interface definitions. However, the `ConvexStakingWrapperSilo` contract itself does not implement the `IWrapperDepositor`. Without this, there are no compile-time guarantees that the contract will be compatible with the calls to the functions that the interface defines. This can lead to potential runtime errors and exceptions that are hard to debug. Explicitly defining that a contract implements an interface could minimize such errors.

Code corrected:

`ConvexSiloWrapper` now implements `IConvexSiloWrapper`.

6.11 A Collateral-Only Silo Can Have Its `siloAsset` Borrowed

Informational Version 1 Code Corrected

During the audit, we uncovered that a silo with a collateral-only asset could still have this asset borrowed. More specifically, this can happen when tokens are sent directly to the silo. The collateral-only funds are not affected but the specification is violated.

```
/// @notice Modification of the Silo where a siloAsset can be deposited
/// only as collateral only asset and can't be borrowed.
```

Code corrected:

`if (_isSiloAsset(_asset)) revert();` has been added to `borrow()` and `borrowFor()` in the specialized Silos for Curve and Convex. However, note that the exact mechanics are out of scope, and if there is alternative way to borrow, the issue could still persist.

6.12 Missing NatSpec

Informational Version 1 Code Corrected

Most functions are provided with documentation. However, `CurvePriceProviderETH` does not have any NatSpec for `NULL_ADDRESS`, `WETH` and `_getCoin()`. Further, NatSpec lacks for `ICurvePriceProvider's getAssetPool()` and `isAssetPoolUint256()` functions. Also, `_getDy()` in the `CurvePriceProvider` is not fully documented.

Code corrected:

The files have been removed.

6.13 Usage of `registryId`

Informational Version 1 Code Corrected

`Pool.registryId` is never used.

Code corrected:

Silo Finance removed the `registryId` as it was unused.

6.14 Variables Visibility

Informational Version 1 Code Corrected

In `CurveLPTokenDetailsBaseCache`, `coins` is `public` and has hence an automatic getter. However, `getCoins()` has the same behavior as the automatic getter and thus there is a double getter for the elements.

In `CurvePriceProvider`, `NOT_FOUND_INDEX` is `public`. However, it is only used for protocol internal logic.

Code corrected:

`getCoins()` was removed. Hence, `coins()` remains the only public getter.

`CurvePriceProvider` has been removed.

6.15 `isMeta` for Crypto Pools

Informational Version 1 Code Corrected

The `isMeta` flag is present for Curve crypto pools. However, note that these pools are not meta pools. The `CurveCryptoSwapRegistryFetcher` can set it to `true`. However, it has no effect.

Code corrected:

Client added a sanity check to ensure the validity of the pools data returned by fetchers.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Balancer Oracles Are Deprecated

Informational Version 1 Acknowledged

During the audit we discovered that the protocol uses the Balancer V2 oracles. However, note that Balancer discourages the usage of their oracles here: https://docs.balancer.fi/products/oracles (snapshot).

Acknowledged:

Silo Finance replied:

```
The Silo team is aware of this
```

7.2 Code With No Effects

Informational Version 1 Acknowledged

The `ConvexStakingWrapper` smart contract contains code without any real effects.

On line 215:

```
if(registeredRewards[_token] == 0){
        ...
}else{
        uint256 index = registeredRewards[_token];
        if(index > 0){...}
}
```

The second `if` will always be executed as `index` will always be greater than 0.

On lines 167, 169, 210: Token transfers to self has no effect.

Acknowledged:

Silo Finance replied:

```
We decided to make minimal changes in external contracts that we inherit.
Refactoring is out of scope of this feature.
```

7.3 Commented Code

Informational Version 1 Acknowledged

The following comment contains a line of commented-out code.

```
// collateralVault = _vault;
```

Removing the line could improve readability.

Acknowledged:

Silo Finance replied:

```
We decided to make minimal changes in external contracts that we inherit. Refactoring is out of scope of this feature.
```

7.4 Gas Inefficiencies

Informational Version 1 Acknowledged

The staking wrapper for convex contains several gas inefficiencies. The following is an incomplete list of examples:

- More state variables could be constants and immutables.
- "Double-initialization" performs some storage writes twice (e.g. setting the owner)
- When adding rewards, the rewards length is always read from storage.
- `registeredRewards` is read twice in the `else` branch of `addTokenReward()`.
- `_calcRewardIntegral` reads variables multiple times from storage (e.g. `reward_remaining`)

Acknowledged:

Silo Finance replied:

```
We decided to make minimal changes in external contracts that we inherit. Gas
optimization in external contracts is out of scope of this feature.
```

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Collateral-only Assets

Note Version 1

Neither the Curve LP tokens nor the wrapped Convex tokens are suitable for borrowing due to their prices being easily but legitimately pushed upwards. Hence, the tokens are only suitable as collateral assets.

8.2 Convex DOS Potential

Note Version 1

In the convex staking wrapper contract, checkpointing could iterate over many tokens. If Convex adds too many reward tokens, the checkpointing could be DOSed. While Convex is trusted in that sense, users should be aware of such a possibility.

8.3 Duplicate Tokens Undervalue Estimation of Rewards

Note Version 1

In the `ConvexStakingWrapper` smart contract, extra reward tokens are queried from the convex reward pool and can also be added manually by the owner.

If there is a case of a pool that receives rewards of the same token from two different reward pools, then only one of these will be queried with the `earned()` to estimate the rewards for this token in the `earnedView()` function of the staking wrapper, leading to low reward estimations.

Further note that any reward token that has no `reward_pool` associated will have its rewards undervalued in the view functions. However, note that Convex is not expected to behave in such a way.

8.4 Function Interface Is Not Validated

Note Version 1

In `CurvePriceProvider`, the manager must provide a `GET_DY_INTERFACE` enum due to the interfaces of the `get_dy()` function being different across curve pools. However, this enum is not sanity checked while most other set-up arguments are and a pool with the wrong interface could be saved to storage.

8.5 Oracle Manipulation

Note Version 1

Note that any on-chain oracle is manipulatable to some degree. Hence, the prices of assets could be manipulable to some extent.

8.6 Read-only Reentrancy Protection

Note Version 1

Users and Silo Finance should be careful when the gas costs of opcodes change in new hardforks as this could lead to breaking changes. Hence, this should be monitored. Further, the selection of parameters should be carried out with gas measurements on the pools since different Vyper versions and pools may need to different parameters.

8.7 The Curve LP Oracles Fetch the Minimum Price

Note Version 1

The Curve lp oracles will fetch a price that is a lower bound on the LP token price. When used as a collateral, users should know that this is the case and be extra careful to avoid unnecessary liquidations.

8.8 Unsupported Reward Tokens

Note Version 1

Users should be aware that reward tokens that can change the amount differently from the amount transferred could unfairly distribute rewards and could break the contract.

Especially, rebasing tokens could break the contract if a rebase downwards occurs. Further, reward tokens with transfer fees could create problems.