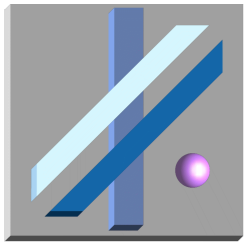

Fixed Point Solutions, LLC

Gro StableConvexXPool Assessment

2022/03/30
Prepared by: Kurt Barry

1. Scope

The StableConvexXPool strategy deposits Curve LP tokens from metapools (which pair a stable coin against 3CRV LP tokens) into Convex to earn yield. It uses one of the underlying 3pool coins (DAI, USDC, or USDT) as its unit of account. The following files (at the linked commit hash) were in-scope:

https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol
https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/BaseStrategy.sol#L491

Other contracts, such as the Convex contract or Gro Vault implementation, were referenced as needed but were not explicitly assessed.

2. Limitations

No assessment can guarantee the absolute safety or security of a software-based system. Further, a system can become unsafe or insecure over time as it and/or its environment evolves. This assessment aimed to discover as many issues and make as many suggestions for improvement as possible within the specified timeframe. Undiscovered issues, even serious ones, may remain. Issues may also exist in components and dependencies not included in the assessment scope.

3. Findings

Findings and recommendations are listed in this section, grouped into broad categories. It is up to the team behind the code to ultimately decide whether the items listed here qualify as issues that need to be fixed, and whether any suggested changes are worth adopting. When a response from the team regarding a finding is available, it is provided.

Findings are given a severity rating based on their likelihood of causing harm in practice and the potential magnitude of their negative impact. Severity is only a rough guideline as to the risk an issue presents, and all issues should be carefully evaluated.

Severity Level Determination		Impact		
		High	Medium	Low
Likelihood	High	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Issues that do not present any quantifiable risk (as is common for issues in the Code Quality category) are given a severity of **Informational**.

3.1 Security and Correctness

Findings that could lead to harmful outcomes or violate the intentions of the system.

SC.1 No Slippage Protection When Selling or Estimating CRV and CVX Rewards

Severity: Low

Code Location:
[1]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L325
[2]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L335
[3]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L236

Description: The direct DEX swaps set the minimum amount of output token to receive to zero, creating a risk of sandwich attacks or other MEV extraction. Full mitigation would require the introduction of trusted oracles; decent countermeasures include using private RPCs and harvesting frequently so the amount of MEV is unattractive to arbitrageurs.

When estimating the total strategy assets, estimation is again done with no slippage limits for reward swaps. This is already somewhat mitigated, however, by the slippage checks that exist for withdrawals.

Response: A private RPC is used to submit transactions that sell rewards, limiting MEV attacks to uncle bandits; also, a high frequency of harvests limits value extraction. Testing has shown that the existing slippage limits for withdrawals are sufficient to protect against manipulation of total asset estimation.

SC.2 Access Control to `harvest()` Disabled For Testing

Severity: Low

Code Location:
https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/BaseStrategy.sol#L491

Description: Having access control on `harvest()` is an important security property given the potential for MEV extraction for any actor that calls it; it has, however, been disabled in the code. According to the team, this is for testing purposes only and the line is un-commented for deployments. At a minimum this should be documented; ideally, access control limitations should be worked around in the tests themselves to avoid the risk of accidentally deploying a vulnerable strategy.

Response: Will add a comment to document this.

SC.3 `adjustPosition()` Ignores Its `_debtOutstanding` Argument

Severity: Low

Code Location:
https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L392

Description: The comment on this function states that the `_debtOutstanding` argument is expected to be zero; while this is almost always true if it is called as part of a harvest (and in case it is non-zero, there will not be free capital to re-invest), this is not guaranteed if it is invoked via `tend()`.

Response: This strategy does not use `tend()`--could override it with an empty function to make this explicit.

SC.4 Older Solidity Version In Use

Severity: Low

Code Location:
https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L2

Description: The code is locked to Solidity version 0.8.4. While wariness of the bleeding edge is often justified with any dependency, the risk of using older versions is that they might have bugs that are fixed in later versions. In particular, there is a near miss here–the code uses a signed, immutable value with width less than 32 bytes (`int128 public immutable WANT_INDEX;` in StableConvexXPool.sol). A bug involving such values was fixed in Solidity 0.8.9:

https://blog.soliditylang.org/2021/09/29/signed-immutables-bug/

Fortunately, the Solidity team has very high confidence that this bug can only manifest incorrect behavior if such a value is accessed via inline assembly, which is not done here. The issue as described also seems to only affect negative values; `WANT_INDEX` should always be positive. It may still, however, be worth considering moving at least to Solidity 0.8.9 (released September 29th, 2021) to eliminate any lingering uncertainty.

Response: Will update to solc 0.8.10 in a future release.

3.2 Usability and Incentives

Findings that could lead to suboptimal user experience, hinder integrations, or lead to undesirable behavioral outcomes.

None.

3.3 Gas Optimizations

Findings that could reduce the gas costs of interacting with the protocol, potentially on an amortized or averaged basis.

G.1 Redundant Events

Severity: **Low**

Code Location:
[1]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L118

[2]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L119

Description: When `switchDex()` is called, both of these events will be emitted with identical arguments and indexing. Only emitting one event would use less gas.

Response: Will implement in a future release.

G.2 Explicit Initializations Are Unnecessary and Add Gas

Severity: **Low**

Code Location:
[1]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/BaseStrategy.sol#L492
[2]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/BaseStrategy.sol#L493
[3]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/BaseStrategy.sol#L495

Description: Solidity starts variables off with a default value of zero, so explicit initialization is unnecessary; further, at least of recent 0.8.X solc versions, the compiler does not yet optimize away explicit initialization to default values.

Response: Will implement in a future release.

G.3 Numeric Operations That Could Use Unchecked Arithmetic

Severity: **Low**

Code Location:
[1]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L527
[2]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L530
[3]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L535
[4]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L537
[5]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L545
[6]https://github.com/groLabs/gro-strategies/blob/83f9e574187d6b2796b9109f7633ef61c034827d/contracts/strategies/StableConvexXPool.sol#L550

Description: Mathematical operations on these lines have their safety guaranteed by nearby conditional statements, and thus could be done without over/underflow checks to save gas.

Response: Will implement in a future release.

3.4 Code Quality

CQ.1 Inconsistent Function Comments

Severity: **Informational**

Code Location: throughout StableConvexXPool.sol

Description: In some cases, function parameters are documented and described with `@param` annotations, in other cases not.

Response: Acknowledged.

4. Notes

This section contains general considerations for interacting with or maintaining the system and various conclusions reached or discoveries made during the course of the assessment. Whereas findings generally represent things for the team to consider changing, notes are more informational and may be helpful to those who intend to interact with the system.

4.1 Pool Migration Risk and Mitigation

The StableConvexXPool strategy implements the ability to migrate assets from one Curve pool to another. This is useful because the highest-yielding pool is likely to change over time. The method in which this is done, however, introduces a risk–high slippage, which could be maliciously orchestrated. A potentially very large, unbalanced withdrawal is done from the old Curve pool as 3CRV tokens, and the 3CRV is converted to the “want” asset (the token that the Vault deposits into the strategy, and the unit of account for gains and losses). Then, the “want” asset is redeposited into 3pool and the resulting 3CRV is deposited into the new pool. If the price of 3CRV is distorted in either underlying pool (expensive in the old or cheap in the new), potentially due to flash loan manipulation, the strategy could be made to suffer a large loss. A manipulation in 3pool would result in a reported loss, but unless debt is called back as part of the migration, the fact that all “want” tokens are re-deposited into 3pool would cancel out the negative effect on total assets.

This risk is mostly mitigated by slippage checks in `_withdrawAll()` and `adjustPosition()` that estimate the total amount of “want” to withdraw, or LP tokens to receive after deposit, based on the virtual price of the old or new pool, respectively. If the slippage on withdrawal or deposit

exceeds the limit set (which defaults to 10 basis points, or 0.1%), then the migration won't succeed. Because the comparison is done "end-to-end" in both cases (the amount of the initial asset is used to derive the amount of the final asset), slippage in 3pool is covered as well. The main risk that arises is that a persistent imbalance in a pool can prevent a migration from occurring–although such exceptional cases would probably require special intervention anyway. Care should be taken to ensure this strategy never represents too large a share of liquidity in the target pool, as this makes the risk of suffering a forced loss due to an imbalance much higher.