Fixed Point Solutions, LLC

# GSquared Assessment

**2022/09/12**
**Prepared by: Kurt Barry**

## 1. Scope

The GSquared codebase represents the second major version of the Gro protocol, containing a number of improvements and simplifications. Smart contract code in the repository was analyzed at the following commit:

a08bf407552b14374a0fbb3364292250d7cb6c82.

Additionally, work to modularize the profits-net-losses (PnL) logic and implement a different PnL distribution based on a fixed rate for the senior tranche was reviewed. Findings and reflections on this logic are not commingled with findings from the primary review and can be found in Note N.1.

Fixes for findings were checked to ensure they addressed the issues raised.

Approximately 21 person-hours of review effort were spent on this assessment.

## 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 FixedTokenCurve Assumes that the Yield Token and Its Underlying Asset Have the Same Decimals

**Severity: Low**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/utils/FixedTokensCurve.sol#L172

**Description**: In this calculation, the value returned by `getYieldToken(_index).convertToAssets(_amount)` will have the same precision as the underlying asset of the associated ERC-4626 vault; `getYieldTokenDecimals(_index)` will return the precision of the 4626 share token itself. While the two are the same in the context of this codebase, this is not fully general as the EIP-4626 standard does not require that these values coincide, only recommends it. Since the comments of the `FixedTokensCurve` contract indicate it is generic to 4626 vaults, it is advisable to either generalize this function (by directly fetching the precision of the underlying asset), or amend the comments to highlight this limitation.

**Response**: Fixed in PR#77.

## SC.2 The `stopLoss()` Function of `ConvexStrategy` Could Be Called Inappropriately

**Severity: Medium**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L789

**Description**: The only check enforced by the `stopLoss()` function is to ensure the caller is a valid keeper. However, it is also only supposed to be called when the `canStopLoss()` function returns `true`. Thus, all keepers must either faithfully obey the signal from `canStopLoss()` or be constrained by a further authorization architecture. This is potentially an attack vector as the `stopLoss()` will trigger `divestAll()` which removes 3CRV from the current metapool with no slippage protection:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L735. If the pool is in an unbalanced state, a keeper could maliciously cause a large loss to the strategy and keep the profit for itself.

**Response**: A separate smart contract will be designated to call the `stopLoss()` function and granted a keeper role on the strategy; it will enforce the check that it is appropriate to trigger the `stopLoss()` function. Active stop loss keepers will then be authorized to call this contract. A similar structure will be used for other keepers with the exception of a highly trusted deployer address.

**FPS Note on Response**: These practices significantly reduce risk, with the residual concerns being either a misconfiguration or a compromise of the deployer address's private key.

## SC.3 One Branch In `realisePnl()` Fails to Repay Debt

**Severity: Low**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L704

**Description**: This branch of the `realisePnl()` function fails to set the `debtRepayment` variable, which means the strategy will fail to pay back excess debt and profit until a different branch of `realisePnl()` is triggered. It is relatively likely that the next harvest will indeed trigger a different branch, and the emergency mode or stop loss functionality could be used to work around a persistent issue. The worst consequence would thus be a delay in appropriately transferring funds to the vault, and possibly some manual repair actions and a strategy upgrade.

**Response**: Fixed in PR#74.

## SC.4 Profit Return Calculation Differs From Intent

**Severity: Low**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L702

**Description**: According to discussions with the Gro team, the intent of the profit return calculation is to only send back whatever profit *wouldn't* be reinvested in the strategy (with the goal of minimizing gas and slippage costs). To match the intent, the calculation should change from
`uint256 profitToRepay = profit * _debtRatio / PERCENTAGE_DECIMAL_FACTOR;`
to
`uint256 profitToRepay = profit * (PERCENTAGE_DECIMAL_FACTOR - _debtRatio) / PERCENTAGE_DECIMAL_FACTOR;`
.

**Response**: Fixed in PR#74.

## SC.5 No Slippage Protection in `invest()` and `divest()` Function of `ConvexStrategy`

**Severity: Medium**

**Code Location**:
[1]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L728
[2]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L741

**Description**: Both of these functions accept any slippage. In the context of harvests triggered by trusted keepers via private or otherwise MEV-protected RPCs, this is of little concern. However, `divest()` in particular is called in some branches of `withdraw()`, which may be called in public contexts as users exit or enter the associated `GVault`. Although it is always possible to add slippage protection at a higher layer, it is worth noting that the `GRouter` contract implements no slippage check for vault withdrawals (nor does the `GVault` itself); instead, the only check is on the conversion of withdrawn 3CRV in stablecoins (see U.1). It may also be worth considering that keepers may become compromised and take advantage of the complete lack of protection (although, as noted in the team's response, too-stringent protection potentially traps users; however, allowing users to select their preferred slippage would seem to address this). Migrations are another context which may enhance the risk of accepting any slippage.

**Response**: Harvests will indeed be protected by private RPCs. In the past, too-stringent slippage protection has prevented users from withdrawing during times of volatility. User-specified slippage enabled for end-to-end actions in PR#89. Slippage protection for non-user operations added in PR#90.

## 3.2 Usability and Incentives

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

### U.1 The `slippage` Parameter Used In Several GRouter Functions May Create Confusion

**Severity: Low**

**Code Location**:
[1]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GRouter.sol#L172
[2]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GRouter.sol#L201
etc…

**Description**: These `slippage` parameters only apply to the final conversion from 3CRV to stablecoins. But it's possible for losses to occur in other steps (withdrawing from GVault and/or Tranche) relative to what the user might have expected, based on e.g. their PWRD balance. There is nothing necessarily wrong with this, and the comments correctly describe the behavior. However, depending on how the UI represents this parameter, it could lead to unexpected outcomes for users, as the usual expectation is that slippage bounds the relative error between initial and final value involved in a transaction. See also SC.5–the lack of end-to-end slippage protection may expose users to losses from sandwich attacks or other MEV techniques.

**Response**: Fixed in PR#89.

## 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 All Fields of the `Strategy` Structure Could Be Packed Into a Single Storage Slot

**Severity**: **Low**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/utils/StrategyQueue.sol#L29

**Description**: If the `uint48` type were used instead of `uint64` for the `next` and `prev` fields, all data would pack into a single storage slot, which should result in gas savings.

**Response**: Fixed in PR#75.

### G.2 Numeric Operations in the `StrategyQueue` Contract Could Be Unchecked

**Severity**: **Low**

**Code Location**: throughout StrategyQueue.sol

**Description**: Most of the operations are simple counter increments that have no practical risk of overflow, and the code could be wrapped in `unchecked` blocks to save gas. Similar optimization opportunities exist in other parts of the codebase as well, but were not explicitly noted in the interest of time efficiency.

**Response**: Acknowledged. May address in later enhancements.

## 3.4 Code Quality

### CQ.1 Incorrect Function Comments in GVault.sol

**Severity**: **Informational**

**Code Location**:
[1]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L131
[2]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L132

**Description**: These comments are duplicated from `getNoOfStrategies()` and do not apply to `getStrategyDebt()`.

**Response**: Fixed in PR#95.

## CQ.2 `_getStrategyTotalAssets()` Is `internal` and Has One Callsite and Could Be Inlined

**Severity**: **Informational**

**Code Location**:
[1]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L819
[2]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L145

**Description**: The `_getStrategyTotalAssets()` function is `internal` and only called by the `getStrategyAssets()` function; its logic could thus be inlined to improve readability (and likely reduce bytecode size, although this depends on the optimizer).

**Response**: Fixed in PR#78.

## CQ.3 Unclear Reference in Comment

**Severity**: **Informational**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L430

**Description**: There is no dev note on the `withdraw()` function, so it is unclear what this comment refers to.

**Response**: Fixed in PR#79.

## CQ.5 Code Duplication in 4626 Preview Functions

**Severity**: **Informational**

**Code Location**:
[1]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L307
[2]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L325
[3]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GVault.sol#L353

**Description**: The code bodies of these functions could be replaced with calls to either `convertToShares()` ([1] and [3]) or `convertToAssets()` ([2]), a pattern already used for `maxWithdraw()`. This reduces code duplication, improving readability and maintainability, although the extra function call overhead will introduce a small gas cost.

**Response**: Fixed in PR#80.

### CQ.6 Unused `internal` Functions In `GTranche`

**Severity**: **Informational**

**Code Location**:
[1]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GTranche.sol#L412
[2]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GTranche.sol#L565

**Description**: These functions are marked `internal` and are unused ([1] actually has an empty implementation, as well), so they can be removed.

**Response**: Fixed in PR#81.

### CQ.7 Inaccurate Comment in GTranche.sol

**Severity**: **Informational**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GTranche.sol#L433

**Description**: This comment states that the related function is called from the `GMigration` contract; this is not the case, and in fact the function is marked `onlyOwner`.

**Response**: Fixed in PR#83.

### CQ.8 Incorrect Function Comments in GRouter.sol

**Severity**: **Informational**

**Code Location**:
[1]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GRouter.sol#L495
[2]https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/GRouter.sol#L595

**Description**: These comments use the term “withdrawn” when “deposited” would be correct. They also incorrectly specify “tranche tokens” when in fact the argument is a quantity of one of the three stablecoins.

**Response**: Fixed in PR#85.

## CQ.9 Commented Signatures of Custom Errors in ConvexStrategy.sol Are Incorrect

**Severity**: **Informational**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L12

**Description**: The commented values do not match the computed selectors of the errors; e.g.:

```
$ cast sig “NotOwner()”
0x30cd7471
```

vs

```
    error NotOwner(); // 0x251c9d63.
```

**Response**: Fixed in PR#87.

## CQ.10 Typos

**Severity**: **Informational**

**Code Location**: Throughout the codebase.

**Description**: Generally, typos in comments and misspellings in variable names make code harder to read and distract from the main content of the logic. Cleaning them up can help integrators, auditors, and anyone else looking into the code understand it faster and with greater confidence, particularly those for whom English is a second language.

**Response**: Acknowledged, will clean up later in the development cycle.

## CQ.11 Obsolete TODO

**Severity**: **Informational**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/ConvexStrategy.sol#L844

**Description**: This TODO seems to be completed and can thus be removed.

**Response**: Fixed in PR#84.

### CQ.12 Incorrect Comment in RouterOracle.sol

**Severity**: **Informational**

**Code Location**:
https://github.com/groLabs/GSquared/blob/a08bf407552b14374a0fbb3364292250d7cb6c82/contracts/oracles/RouterOracle.sol#L109

**Description**: The comments preceding the `getAggregator()` function do not accurately describe what it does.

**Response**: Fixed in PR#82.

# 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.

## N.1 Review of Modularized Profits-Net-Losses (PnL) Logic and Fixed Rate for Senior Tranche PnL Implementation

### Modularized PnL Logic

https://github.com/groLabs/GSquared/pull/55

This PR fully compartmentalizes all notions of profit sharing and profit/loss away from GTranche and into the PnL logic. The code duplication in the tranche seems to be a necessary consequence of this, although if a way can be found to eliminate it without breaking the PnL abstraction, that would be preferable.

### Fixed Rate for Senior Tranche PnL Implementation

https://github.com/groLabs/GSquared/pull/56
This PnL logic eliminates the notion of a “junior loss” that must be reset or earned away before the senior tranche can receive yield again. Instead, a fixed rate is paid to the senior tranche, dipping into junior tranche assets if necessary. This essentially replaces a rare need for the DAO to take controversial action by resetting a junior loss with a more frequent but less charged

decision to adjust the fixed rate paid to the senior tranche (which will generally need to be kept below the rate at which interest is earned by the underlying yield token, adjusted for utilization).

One risk is that if the rate is not updated promptly to keep it in line with market rates, the protocol may experience large TVL fluctuations, especially in the senior tranche.

This model also adds a further significant risk for the junior tranche. Since junior tranche depositors cannot withdraw when the utilization (the ratio of senior tranche assets to junior tranche assets) is too high, the junior tranche can become “trapped” if the utilization is high and the fixed rate exceeds the floating rate of the underlying yield token for a significant period of time. While the senior tranche stops earning yield if the utilization goes above unity, senior depositors can simply continue to make small withdrawals to keep the junior depositors trapped. So long as the Gro DAO is diligent about adjusting the fixed rate (or better yet finds a way to automate setting it), this is easy to mitigate. If for any reason the Gro DAO ceases to operate, or becomes incentive-misalinged (for example, controlled primarily by senior tranche depositors), the junior tranche depositors could have their funds drained entirely.