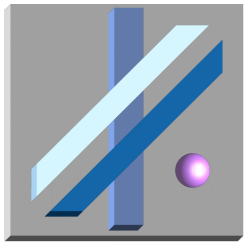

Fixed Point Solutions, LLC

Gro Protocol Assessment

2021/07/20
Prepared by: Kurt Barry

Scope

Gro is a stablecoin yield aggregation protocol that creates two separate risk tranches for depositors. The senior tranche (represented by a rebasing token called PWRD) receives a lower proportional share of profits but is insulated against loss by the junior tranche (represented by a value-accumulating token called GVT). The protocol invests deposited stablecoins across a range of strategies, aiming to diversify and balance risk exposures to both the underlying stablecoins and yield-generating strategies. It makes heavy use of Curve's 3pool for stablecoin exchange and pricing.

Review was performed on the "develop" branch of the core Gro protocol repository (https://github.com/groLabs/gro-protocol) over two separate engagement periods, for roughly 50 total person hours of effort. The repository was under active development during the review so multiple commit hashes were examined. The deposit and withdrawal flows were covered in detail. The Vaults (contained in gro-protocol/contracts/vaults) were left out-of-scope. Interactions with Curve's 3pool were in-scope although the Curve protocol itself was out-of-scope.

Earliest commit examined: 438e29366368e4ec0a8be3ae39ec1d369db10991
Latest commit examined: 634a0ee339001674e56e04294c9c63f195955c46

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 an issue is available, it is provided. No guarantee is made that the findings here are exhaustive.

Security and Correctness

SC.1 - Inaccurate Pricing If Stablecoins Lose Their Pegs

https://github.com/groLabs/gro-protocol/blob/634a0ee339001674e56e04294c9c63f195955c46/contracts/pools/oracle/Buoy3Pool.sol#L171

Team Response: The delay provided by the health check ensures the issue is real and allows time for bots to react and trigger the emergency handler logic. If the deviation is not severe, the pricing error should be tolerable.

Description: Curve's `get_virtual_price` is only an accurate representation of the dollar value of an LP token if none of the underlying stablecoins significantly deviate from their peg. The health check, which compares Curve prices against previously cached Curve prices (which were themselves checked against ChainLink prices), only temporarily protects the system, until the cached prices catch up to the market prices (since a real depegging event should be reflected in both Curve and ChainLink prices). If the emergency logic is not triggered in time, or if the threshold of deviation required for triggering it is too large, this mispricing may have unexpected consequences.

SC.2 - Losses Are Front-Runnable

Team Response: Acceptable risk--only applies in extreme events, and someone would need to build their own bot to detect the incoming loss and execute a frontrunning attack. The known alternatives all greatly increase gas costs or come with other risks (e.g. small depositors still being able to front-run).

Description: With the architecture change from executing to PnL calculations on large deposits and withdraws to instead performing periodic asynchronous updates based on strategy harvests, the door is opened for users to front-run large unrealized loss events. In particular, since GVT holders bear the brunt of any loss, they are strongly incentivized to flee the system in advance of a loss becoming properly accounted for.

SC.3 - Incorrect Rounding in `GToken`

https://github.com/groLabs/gro-protocol/blob/develop/contracts/tokens/GToken.sol#L59

Team Response: The logic eliminates small inconsistencies when depositing and immediately withdrawing the full amount. The occasional 1-wei errors in the `base == false` case are not consequential enough to be worth the extra gas to fix.

Description: The rounding applied here is only correct in the `base == true` case; in the `base == false` case, the correct condition for rounding up is `diff >= b / 2`.

SC.4 - Controller Stablecoin Balances Included In Its Total Assets

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/Controller.sol#L255

Team Response: Fixed.

Description: The `_totalAssets()` function includes the stablecoin balances of the `Controller` in its calculation; however, there seems to be no way for such assets to be transferred out of the `Controller`, and the function level comment doesn't mention that the contract's own balances should be included. The protocol's accounting could be corrupted by sending tokens directly to the `Controller`.

SC.5 - Curve Pool Interactions Without Slippage Checks

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L121
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L415
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L419

Team Response: The health check (a comparison of current Curve ratios against the last "known-good" values which were verified against ChainLink prices) provides a strong protection against manipulation even in the absence of the EOA-only restriction. The user-defined slippage protection also allows users to protect themselves (although this does not protect the protocol).

Description: No slippage checks are performed in these code locations where deposits into or withdrawals from the Curve pool occur. There are no obvious vulnerabilities here, particularly while flash loans/contract callers are prevented and withdrawal fees are non-zero. It also appears that there are no "hookable" external calls (i.e. calls that a caller can affect the behavior of) within the codebase (at least, none within the examined scope). However, lack of slippage checks is a common factor in many exploits, and these areas should be revisited each time the protocol is modified to ensure they do not become problematic (e.g. if the flash loan prevention safeguard is removed or disabled, and especially if any hookable external calls are added, as the latter would bypass the healthcheck). There could also potentially be interactions between these "unlimited slippage" operations and the mispricing noted in SC.1 in a depegging scenario.

SC.6 - Inconsistently Enforced Minimum Asset Value

https://github.com/groLabs/gro-protocol/blob/aaf8a1fd2b734bd731b37428390c54c2ad8f05b1/contracts/pnl/PnL.sol#L394

Team Response: Fixed.

Description: The other branch of the conditional enforces a minimum value for gvtAssets, but this branch does not--in particular, if loss == lastGvtAssets, gvtAssets will go to zero.

SC.7 - Profits From Curve Vault Not Accounted For

https://github.com/groLabs/gro-protocol/blob/41aaf235421dcd92715a0970ae3f79612bb45c36/contracts/pnl/PnL.sol#L278

Team Response: Fixed.

Description: This function neglects profits from the Curve Vault.

SC.8 - Unrestricted Disablement of Flash Loan Prevention

https://github.com/groLabs/gro-protocol/blob/ca9aff7a46c646d28836c2dc595404d9a71ce510/contracts/Controller.sol#L156

Team Response: Fixed.

Description: Any address can turn off the flash loan prevention check for any other address.

SC.9 - Unrestricted Setting of Governance Parameter

https://github.com/groLabs/gro-protocol/blob/438e29366368e4ec0a8be3ae39ec1d369db10991/contracts/pools/LifeGuard3Pool.sol#L102

Team Response: Fixed.

Description: Any address can set the investToCurveThreshold governance parameter.

SC.10 - Unrestricted Disablement of Health Check

https://github.com/groLabs/gro-protocol/blob/438e29366368e4ec0a8be3ae39ec1d369db10991/contracts/pools/LifeGuard3Pool.sol#L109

Team Response: Fixed.

Description: Any address can disable the Curve pool health check.

Gas Optimizations

G.1 - `LifeGuard3Pool` Storage Variables Could Be `immutable`

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L34

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L35
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L36

Team Response: Fixed.

Description: The `crv3pool`, `lpToken`, and `buoy` storage variables in the `LifeGuard3Pool` contract cannot be changed, and could thus be made `immutable` to reduce gas costs.

G.2 - `Buoy3Pool` Storage Variables Could Be `immutable`

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/oracle/Buoy3Pool.sol#L34
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/oracle/Buoy3Pool.sol#L35
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/oracle/Buoy3Pool.sol#L36

Team Response: Fixed.

Description: In the `Buoy3Pool` contract, the `curvePool` and `lpToken` storage variables are mutable, but since changing these would necessitate replacing the `LifeGuard3Pool` contract, they could likely be made `immutable` as well. The `chainOracle` storage variable is only assigned in the constructor, and therefore should be `immutable`.

G.3 - Conditionals Eliminate Need for SafeMath

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pnl/PnL.sol#L398
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pnl/PnL.sol#L400

Team Response: Fixed.

Description: On these two lines, the conditionals guarantee the math is safe, so unchecked math operations could be used, saving some gas.

G.4 - Unnecessary SafeMath

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/oracle/Buoy3Pool.sol#L75

Team Response: Fixed.

Description: Unnecessary SafeMath (`i.add(1)` will always be safe as `i` will be small).

Comments

C.1 - Comment Should Say “mint” Instead Of “burn”

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/DepositHandler.sol#L271

Team Response: Fixed.

Description: Inaccurate comment--should say “mint” instead of “burn”.

C.2 - Unresolved TODOs

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L436
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/WithdrawHandler.sol#L134
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/EmergencyHandler.sol#L137

Team Response: Fixed.

Miscellaneous

M.1 - Unnecessary Local Variable in `GToken`

https://github.com/groLabs/gro-protocol/blob/develop/contracts/tokens/GToken.sol#L50

Team Response: Acknowledged.

Description: No gas savings results from storing `BASE` in the `_BASE` stack variable because `BASE` is already constant (and thus won't be in storage: https://docs.soliditylang.org/en/latest/contracts.html#constants). This local variable could be removed, simplifying the code.

M.2 - `emergencyWithdrawal` Function is `view` and `onlyWhitelist`

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/pools/LifeGuard3Pool.sol#L382

Team Response: Fixed.

Description: This function is both a `view` and `onlyWhitelist`--the modifier may be able to be removed (it won't prevent the data being read off-chain, and there isn't a clear reason to limit access to this function on-chain--`buoy.getRatio()` is `public` anyway). The name of the function seems suboptimal as well--something like `getEmergencyPrice` would be more self-documenting.

M.3 - Redundant Code in Controller

https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/Controller.sol#L120
https://github.com/groLabs/gro-protocol/blob/199b561eb19e0c4608784cfff1f7d6d089097f33/contracts/Controller.sol#L124

Team Response: Needed to conform to interfaces for other contracts.

Description: These functions are redundant since `underlyingTokens` is a public variable. Either these can be removed, or `underlyingTokens` made private, to reduce bytecode size and code complexity.