yAudit Rage Trade Review

Review Resources:

- Rage Trade Docs

Auditors:

- Jackson
- engn33r
- Invader-tak

Table of Contents

b. Invader-tak

Review Summary

Rage Trade

Rage Trade provides two vaults, one risk-on and one risk-off, which allow users to deposit into and earn yield. The risk-off vault earns yield from lending USDC on Aave and a fraction of the risk-on vault rewards. The risk-on vault earns rewards by providing delta neutral liquidity to GMX, hedging the ETH and BTC exposure with short positions on Aave and Uniswap. The risk-on vault acts as a junior tranche and borrows from the risk-off vault (senior tranche) to gain leverage.

The contracts of the Rage Trade Repo were reviewed over 3 weeks. The code review was performed by 3 auditors between December 19, 2022 and January 16, 2023. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit a2107d37b789494454bd4ede7d217d8723474de4 for the Rage Trade repo.

Scope

The scope of the review consisted of all the contracts in the repo at the specific commit. The senior tranche makes reference to a `leveragePool` address that is currently left empty for future use, and therefore it was out of scope. After the findings were presented to the Rage Trade team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudits and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Rage Trade and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Average	Several access control modifiers for privileged roles exist, including `onlyOwner`, `onlyKeeper`, `onlyBorrower`, `onlyDnGmxJuniorVault` and `onlyBalancerVault`. While this is standard for such a project, the compromise of any account with a privileged role can be problematic, so keeping these roles to a minimum is preferable.
Mathematics	Average	No specialized algorithms or custom math operations are needed for Rage Trade, but there are a substantial number of internal accounting values that are used. The math backing the delta neutral strategy adds some complexity, and some values like the senior vault leverage percentage are not hard coded but rather implied based on the process of borrowing and lending that takes place. However, such math and complexity is not easily avoided for a delta neutral strategy.
Complexity	Low	There is significant complexity introduced by the many tokens swaps, the different decimals, the way the contracts are organized, and the implied assumptions in the design that were found with backtesting. Refactoring the code in places to simplify it would assist with future security reviews and may improve the overall health of the protocol so other developers can more easily onboard and understand the design.
Libraries	Average	Math libraries from Uniswap, Aave, and Rari are used. Openzeppelin libraries allow for pausing contracts and transferring ownership. This is a standard number of external dependencies, but relying on external dependencies can introduce additional complexity to a project. This case the complexity of three different math libraries is worth noting.
Decentralization	Average	Different roles have different privilege levels, and the use of a proxy adds to the power that the owner has to make

Category	Mark	Description
		changes. However, this is relatively standard for a protocol that is not aiming for an immutable deployment.
Code stability	Average	Rage Trade has launched on mainnet, but some new upgrades will be completed in January after this review is completed. Because the deployed contracts use proxies, upgrades are not too difficult.
Documentation	Average	There is consistent NatSpec in the code, but the official docs online are lacking in providing an overview of the big picture of how the strategy works. For example, there is no explanation how the 2.3x leverage number for the senior vault is achieved, how the rewards get converted after depositing into GMX, and how the reward split between the senior tranche and junior tranche is calculated.
Monitoring	Average	Most functions that alter state variables emit events, but some do not. For example, `updateFeeStrategyParams()` in DnGmxSeniorVault and some onlyOwner functions in DnGmxJuniorVault like `unstakeAndVestEsGmx()`, `stopVestAndStakeEsGmx()`, and `claimVestedGmx()`.
Testing and verification	Average	The tests maintain good coverage and all tests pass. The test coverage is over 80% for the main vault contracts and libraries. The tests even check for strategy performance at different asset price points to test for adverse market conditions. Test coverage for DepositPeriphery and WithdrawPeriphery is low, below 50%, and should be improved.

Fuzzing

Areas of Focus: Fuzzing campaigns were run to ensure that the expected ERC4626 properties were met, as well as making sure the vaults code did not revert in unexpected circumstances. The results of the fuzzing campaigns can be seen below. Note that some of the campaigns need more work to establish if the scenario passes (the result is marked as “still testing” in these cases).

Contracts Tested: DnGmxJuniorVault.sol (J), DnGmxSeniorVault.sol (S), DepositPeriphery.sol (D), WithdrawPeriphery.sol (W)

Applicable Standards: EIP-4626 (J,S)

Contract	Property	Result
J, S	TotalAssets will never revert	Passed
J, S	ConvertToAssets will never revert*	Passed
J, S	ConvertToShares will never revert*	Passed
J, S	Division rounding errors always favors the protocol	Passed
J, S	Withdraw will never revert**	Passed
J, S	Redeem will never revert**	Passed
J, S	Price per share will never go down	Still testing
J, D, W	Manipulating Uniswap pools to decrease price of WETH/WBTC should not favour user deposit/withdrawals	Passed
J, D, W	Manipulating Uniswap pools to increase price of WETH/WBTC should not favour user deposit/withdrawals	Still testing
J, D, W	Natural price decrease of WETH/WBTC should not should not favour user deposit/withdrawals	Still testing
J, D, W	Natural price increase of WETH/WBTC should not should not favour user deposit/withdrawals	Still testing

* Given that preconditions have been met

** There are a lot of preconditions that needs to be satisfied when a deposit/withdrawal is being executed against one of the delta neutral vaults or the periphery contracts, some of them outside of the users control (oracle price/slippage during flash loan to balance the hedge etc.). These fuzzing campaigns attempted to control for these factors to see if any scenario could occur where a withdrawal/redeem was expected to be possible but didn't.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Critical Findings

None.

High Findings

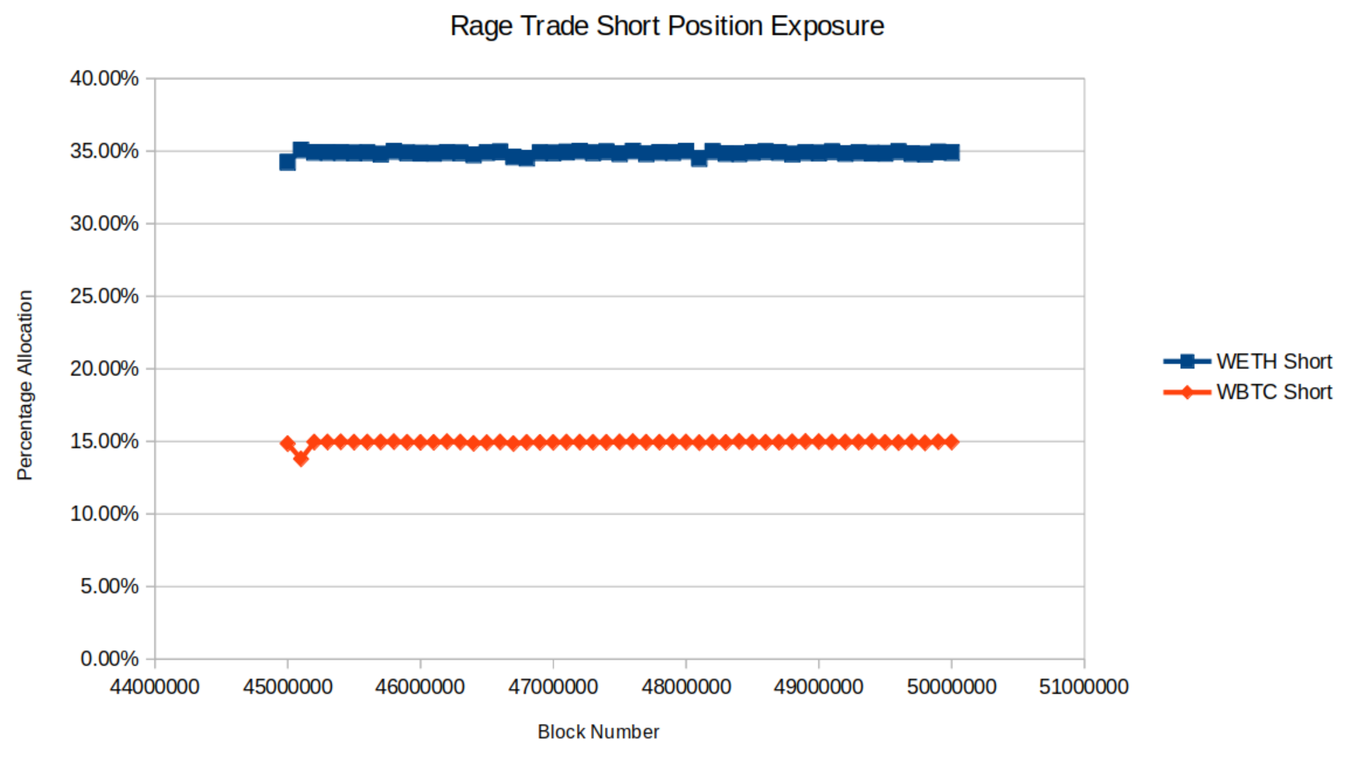
1. High - Incorrect ratios used for delta neutral rebalancing

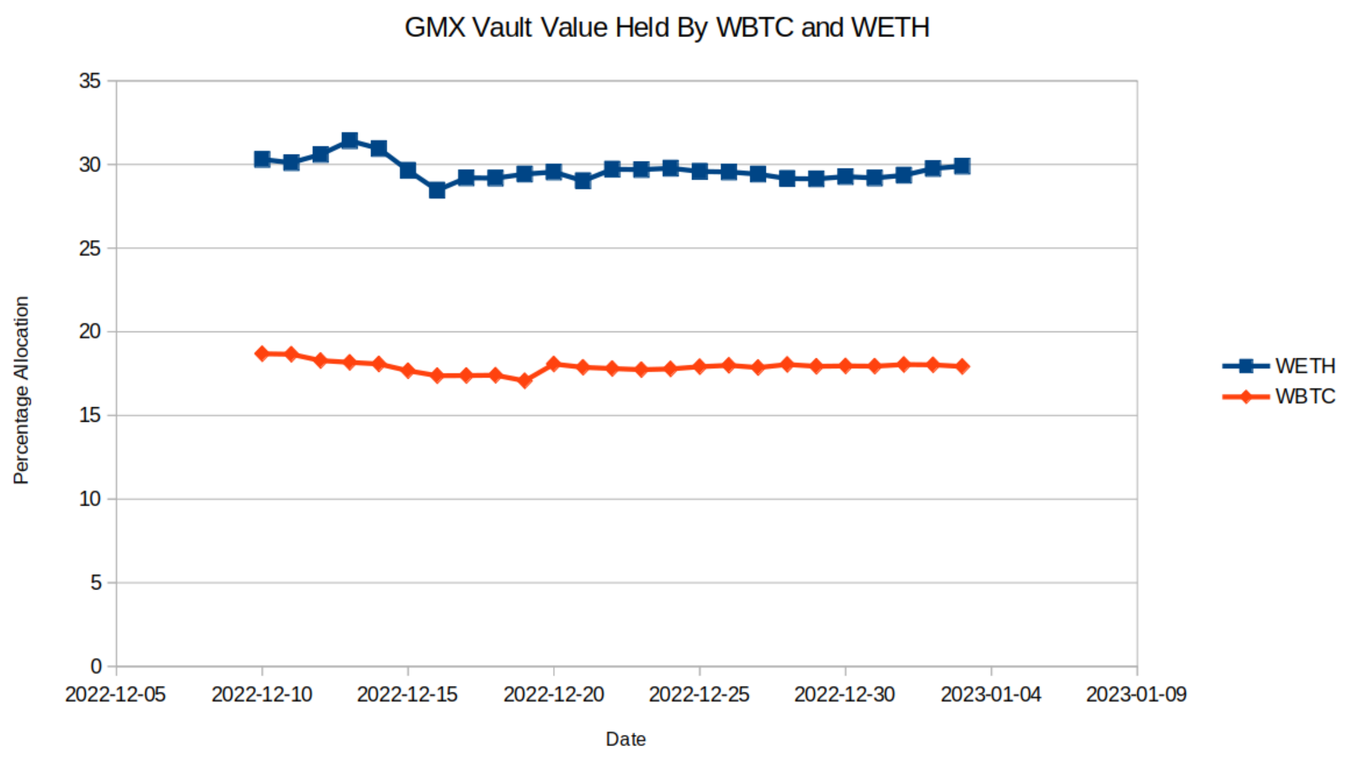
The goal of the Rage Trade Delta Neutral strategy is to maintain a delta neutral position to avoid exposure to impermanent loss. The long and short positions in the strategy are not balanced, meaning the strategy is not delta neutral.

Technical Details

Rage Trade relies on providing liquidity to the GMX protocol for long exposure to WETH and WBTC. The GMX Vault is where all GMX protocol underlying assets are stored. The GMX vault has a target weight for each underlying asset, but the target weight is not the same as the current weight of the asset in the vault during a specific block. The actual weights of the tokens The Rage Trade rebalancing approach uses the ideal target weight, not the actual current weight of the asset. This means the Rage trade strategy is not actually delta neutral because it assumes an ideal mix of underlying assets when holding GLP and doesn't consider the current underlying assets.

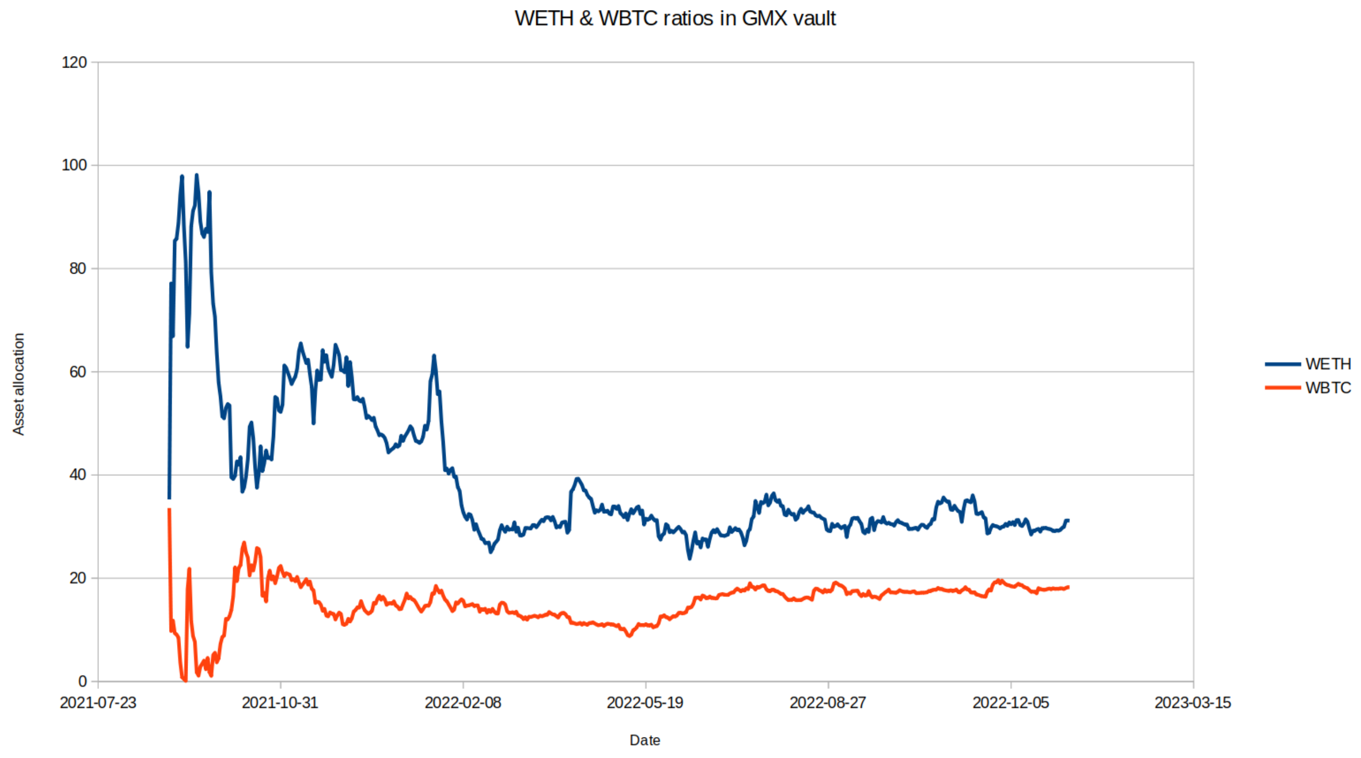
To demonstrate the issue, data was taken from the Arbitrum blockchain to plot the exposure that Rage Trade has had to the underlying assets WETH and WBTC since around the time that the delta neutral strategy began receiving funds. The short exposure takes the form of WETH or WBTC borrowed from Aave. To calculate this percentage, the amount of WETH or WBTC borrowed was multiplied by the price of the asset and then divided by the total value of GLP held by the Rage trade strategy. The short exposure does a good job of targeting the 35% and 15% marks for WETH and WBTC respectively. The 35% and 15% marks are the target allocations for the GMX vault to hold of WETH and WBTC. The long exposure takes the form of GLP, which holds roughly 50% stablecoins and 50% WETH and WBTC. The data for the exposure to WETH and WBTC was taken from at stats.gmx.io during the same period as the short position exposure chart. The comparison of the short and long exposure to WETH and WBTC is shown below.

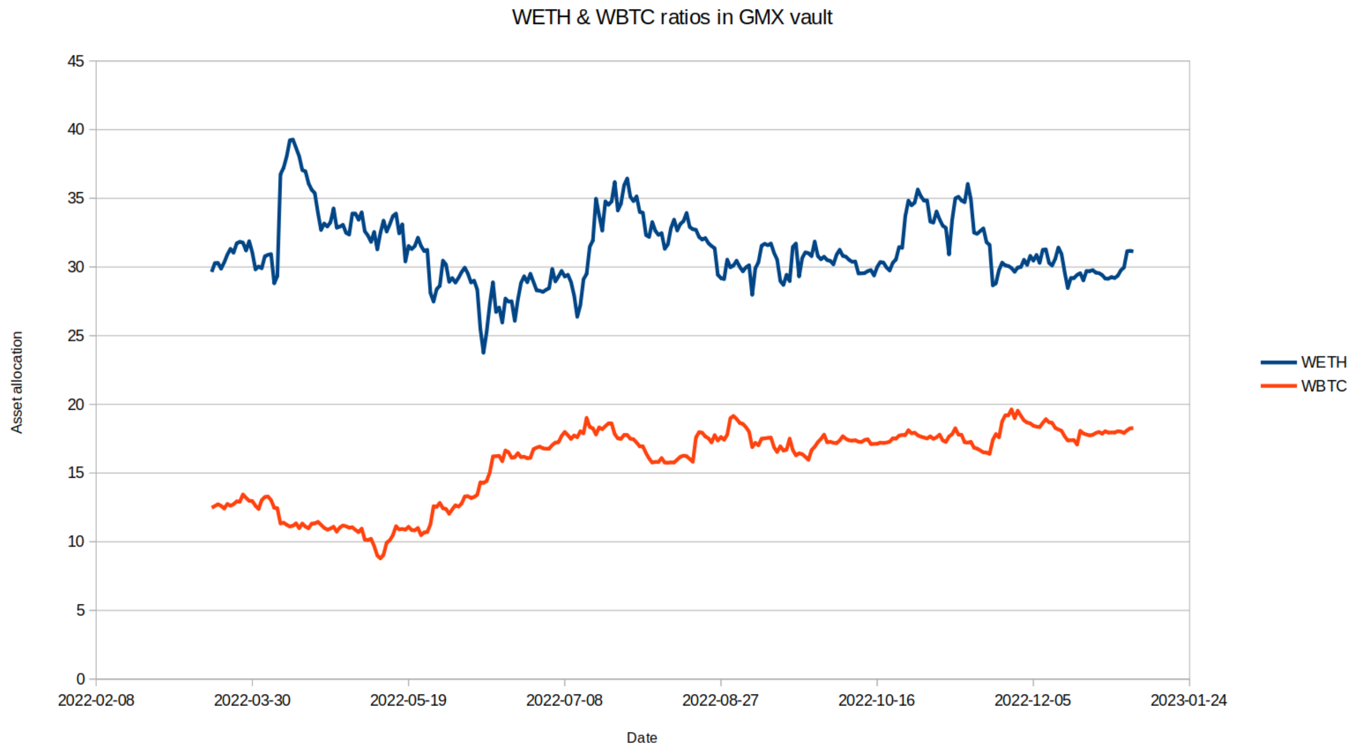




As the charts demonstrate, since the deployment of Rage Trade in December 2022, the GMX vault, and therefore GLP, has consistently held more WBTC and less WETH than the GMX vault target weights. But the Rage Trade short positions do not reflect this, and instead use the target weights to determine the ideal short exposure to these assets. In its current form, Rage Trade has held a position of long WBTC and short WETH since it was deployed.

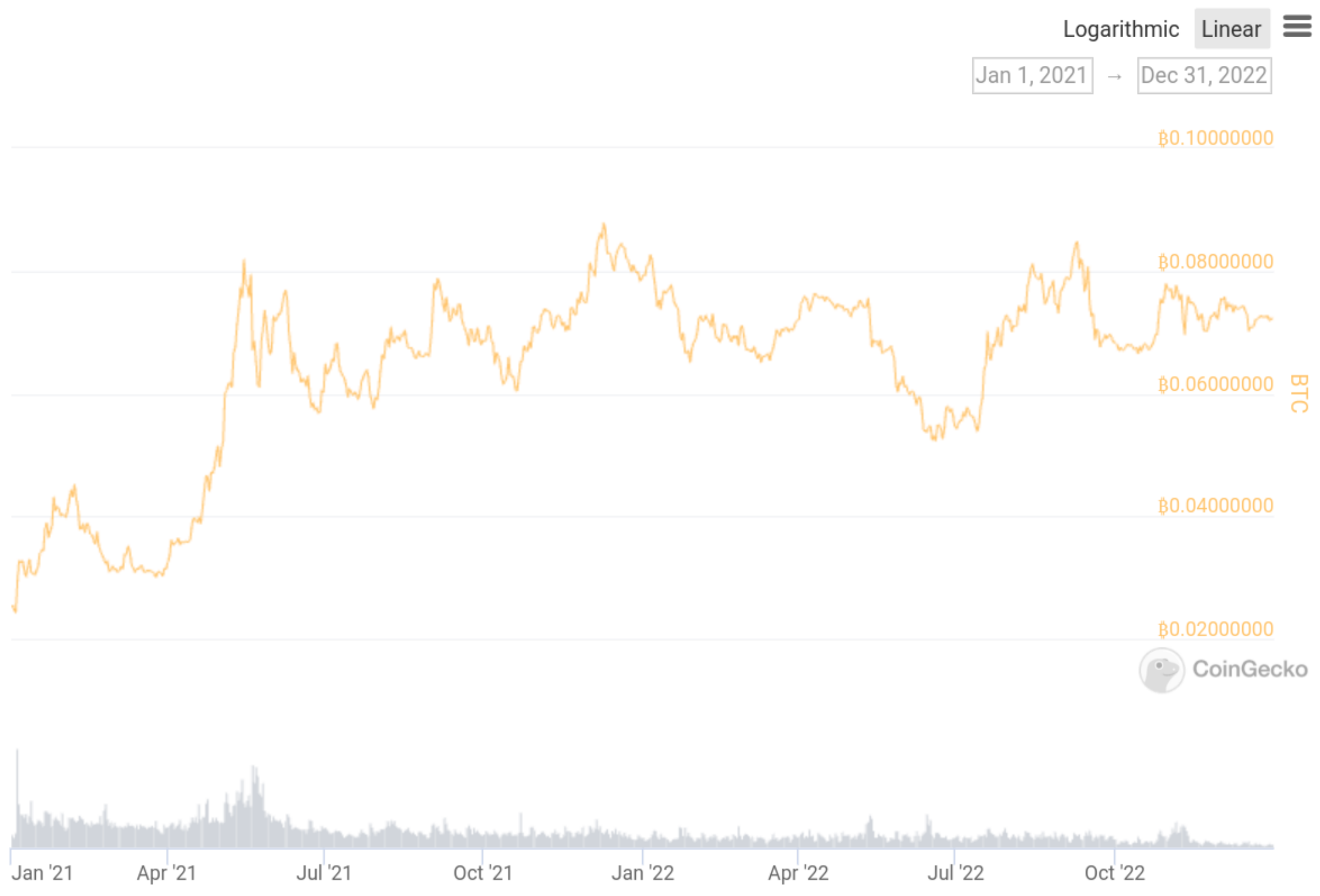
Examining the asset exposure in the GMX Vault over the entire duration of GMX's history, it is more apparent how far the asset allocations can stray from the target. The second figure below focuses on the less volatile period of GMX asset allocation in 2022 to provide a closer look at how different the asset allocation is from the 35% WETH and 15% WBTC targets. Since June 2022, the GMX vault has mostly been overweight in WBTC and underweight in WETH, but sometimes it is overweight in both assets.





Focusing on more “extreme” changes in the asset allocations of the vault, the vault held a total of 39.4% in WETH and WBTC combined on May 26 2022, but that rose to 54% on July 24 2022, less than 2 months later. If this strategy was active at this time, it would have had too large of a short position and too small of a long position. It is true that during this time the value of WETH and WBTC dropped by around 22%, so in actuality these months would have been profitable for this strategy, but a reverse scenario where the assets increased by 22% in value would have left the strategy at a loss.

Even if the above scenario where the strategy is weighted in the wrong direction of market movements does not occur, another risk is if the correlation between WETH and WBTC prices change significantly. It's possible there is an implied assumption that WETH and WBTC prices are correlated, so as long as the sum of allocations WETH and WBTC in the GMX vault is roughly 50% of the GMX vault, then the long position should neutralize the short position. But if the WETH/WBTC price correlation changes while they GMX vault allocations are not near the targets, this could be problematic. The chart from CoinGecko shows this ratio over the last two years and demonstrates that it can easily change by 10% or more in a short timespan.



In summary, the risk this issue highlights includes:

1 strategy losses from when WETH and WBTC positions are overweighted towards a long position and the prices of these assets drops. The reverse can also happen, where

WETH and WBTC positions are underweighted towards a long position and the prices of these assets rises.

2. strategy losses from when WETH and WBTC weights in the GMX vault are not at the target weights and the WETH/WBTC price ratio changes in the wrong direction from how the GMX vault (and therefore the Rage Trade strategy) is weighted.

Impact

High. The strategy does not properly calculate the exposure necessary to maintain a delta neutral position, which leaves depositors vulnerable to losses that can vary depending on market conditions and GMX vault weights.

Recommendation

A few improvements can be made:

1. At the core, the mismatch in long and short positions stems from using `tokenWeights()` to determine the short exposure needed to balance out the GLP holding of WETH and WBTC. `tokenWeights()` is the ideal weighting, not the current allocation to WETH and WBTC, so the current allocation should be used instead.
2. The reweighting process happens at a certain time interval. This result in miniscule changes in asset allocation, as shown in the plot in issue Low 4, which cause allocation changes that are almost irrelevant compared to the mismatch between the target GMX vault weights and the current weights. Instead of using a timer to trigger rebalancing, it would be better to set a threshold related to how unbalanced the positions are. For example, if the difference between the long and short WETH (or WBTC) positions exceeds 0.5% of the GLP value held, then the positions should be rebalanced. This would keep the strategy more neutral with the current GMX vault weights, and even prevent a scenario where the WETH and WBTC prices and vault weights change significantly right after a Rage Trade rebalance happens, which would cause a roughly 12 hour wait time before the next rebalance occurs. The threshold value can be set based on backtesting to account for swap fees and other factors.

Developer Response

Fixed, junior vault now hedges current weights and partial trader PnL.

Medium Findings

1. Medium - Junior vault `getPriceX128()` has wrong decimals

The junior vault's `getPriceX128()` returns a value with an incorrect number of decimals.

Technical Details

The term X128 as used in Uniswap documentation indicates a Q128 value. This means that the value should be divided by `1 << 128` to get the actual value that it represents.

Examining `getPriceX128()` in the junior vault finds a problem. The return value is `aum.mulDiv(1 << 128, totalSupply * 1e24)`. This is similar to the previous junior vault example. `aum` is 1E12, but the denominator is `totalSupply` of 1E18 multiplied by 1E24. This leaves the numerator with 1E12 but the denominator with 1E42, the resulting values is not correct. When the value of `getPriceX128()` in the existing on-chain deployment is divided by `1 << 128`, the resulting value is `8.34E-13`. In contrast, the price of GLP is `$0.834` and the return value of `getPrice(bool)` in the junior vault with 18 decimals like GLP is `0.834E18`.

We can apply the same approach to `getPriceX128()` in the senior vault and find that it is implemented correctly. We can double check the decimals of the return value `price.mulDiv(1 << 128, 1e8)`. `price` is 1E8 and it is divided by 1E8, so the price value is properly multiplied by `1 << 128`.

We can also confirm the decimals in `getPrice(bool)` in the junior vault are correct. The price value is documented in GMX docs:

> The sell price would be getAum(false) / glpSupply

`getPrice(bool)` returns `aum.mulDivDown(PRICE_PRECISION, totalSupply * 1e24)`. `aum` is 1E12, `PRICE_PRECISION` is 1E30, `totalSupply` is 1E18 and it is multiplied by 1E24. This results in `1E12 * 1E30 / (1E18 * 1E24) = 1E42 / 1E42 = 1`. And because GLP has a decimals value of 18, the value is returned with 18 decimals. The return value can be confirmed against the GLP price in the GMX frontend.

Impact

Medium. `getPriceX128()` in the junior vault is not implemented correctly and returns a value with incorrect decimals. While `getPriceX128()` in the senior vault is intended only for frontend use, the junior vault does not document the intended use of this function.

Recommendation

Change the return value of `getPriceX128()` in the junior vault to return a value with correct decimals.

```diff
- return aum.mulDiv(1 << 128, totalSupply * 1e24);
+ return aum.mulDiv(1 << 128, 1e12);
```

Developer Response

Here `getPriceX128` returns the price of 1wei GLP when divided by 1«128 in usdc. So that when glp balance is multiplied by price and divided by 1«128 we get the value. Here `aum` is in `1e30`, `totalSupply` is in `1e18` and `glp` is in `1e18` and `usdc` in `1e6`. Hence 18+6/(18+30) since price * glpBalance should give usdc value.

for e.g

```
current getPriceX128 => 327756966734451170974722619
getPriceX128 / (1 << 128) => 9.631911570974895e-13 (which is for 1 wei of glp)


so, 1e18 wei of GLP (1 GLP) = 963191.1570974895 (in least divisible unit of USDC)
```

Low Findings

1. Low - `maxWithdraw()` and `maxRedeem()` are inaccurate if Aave pool doesn't permit withdrawals

If an Aave pool is paused or made inactive, `maxWithdraw()` and `maxRedeem()` will not return the correct amount of tokens that can be redeemed.

Technical Details

EIP4626 states the following requirements for `maxRedeem()`:

> Maximum amount of Vault shares that can be redeemed from the owner balance in the Vault, through a redeem call.
>
> MUST return the maximum amount of shares that could be transferred from owner through redeem and not cause a revert, which MUST NOT be higher than the actual maximum that would be accepted (it should underestimate if

> necessary).
>
> MUST factor in both global and user-specific limits, like if redemption is entirely disabled (even temporarily) it MUST return 0.

If an Aave pool is paused or inactive, `maxWithdraw()` and `maxRedeem()` in the senior vault should return zero. The functions currently do not. Currently the senior vault is not compliant with the ERC4626 spec if the Aave pool is inactive or paused. `maxMint()` and `maxDeposit()` should also have a similar change.

Impact

Low. `maxWithdraw()` and `maxRedeem()` will return a non-zero value even if the pool is paused or inactive and cannot be withdrawn from.

Recommendation

Consider implementing a check in `maxWithdraw()` like this one that checks if the pool is inactive or paused and returns zero if so.

Developer Response

Acknowledged, not fixing. redemption of shares is not disabled by the senior vault itself in this case but the downstream dependent protocols which vault has no direct control of.

2. Low - `maxMint()` and `maxDeposit()` are inaccurate if Aave pool is frozen

A frozen Aave pool does not allow any new supply, which would prevent depositing and minting actions of the senior vault.

Technical Details

EIP4626 states the following requirements for `maxMint()`:

> MUST return the maximum amount of shares mint would allow to be deposited to receiver and not cause a revert, which MUST NOT be higher than the actual maximum that would be accepted (it should underestimate if necessary). This assumes that the user has infinite assets, i.e. MUST NOT rely on balanceOf of asset.
>
> MUST factor in both global and user-specific limits, like if mints are entirely disabled (even temporarily) it MUST return 0.

If an Aave pool is frozen, Aave documentation states it will not accept deposits. Therefore `maxMint()` and `maxDeposit()` should return zero in this case. The functions currently do not. Currently the senior vault is not compliant with the ERC4626 spec if the Aave pool is frozen.

Impact

Low. `maxMint()` and `maxDeposit()` will return a non-zero value even if the pool is frozen and cannot be deposited to.

Recommendation

Consider implementing a check in `maxMint()` and `maxDeposit()` like this one that checks if the pool is frozen and returns zero if so.

Developer Response

Acknowledged, not fixing. minting of shares is not disabled by the senior vault itself in this case but the downstream dependent protocols which vault has no direct control of.

3. Low - `setGmxParams()` doesn't update dependent variable

`setGmxParams()` allows `state.glpManager` to be modified. However, it fails to update values that are dependent on `state.glpManager`.

Technical Details

The value `state.glpManager` can be updated in `setGmxParams()` by the owner. The value `state.gmxVault` is set in `initialize()` by calling `vault()` in `state.glpManager`. When `state.glpManager` is updated, the value `state.gmxVault` is not updated, which may leave the gmx vault variable outdated.

Impact

Low. `setGmxParams()` doesn't change variables that are dependent on the glpManager.

Recommendation

Consider adding this line to `setGmxParams()`:

```
state.gmxVault = IVault(state.glpManager.vault());
```

Developer Response

`gmxVault` is not an upgradable contract, so any glpManager address set via `setGmxParams` will use same canonical underlying `gmxVault` with which junior vault was initialized.

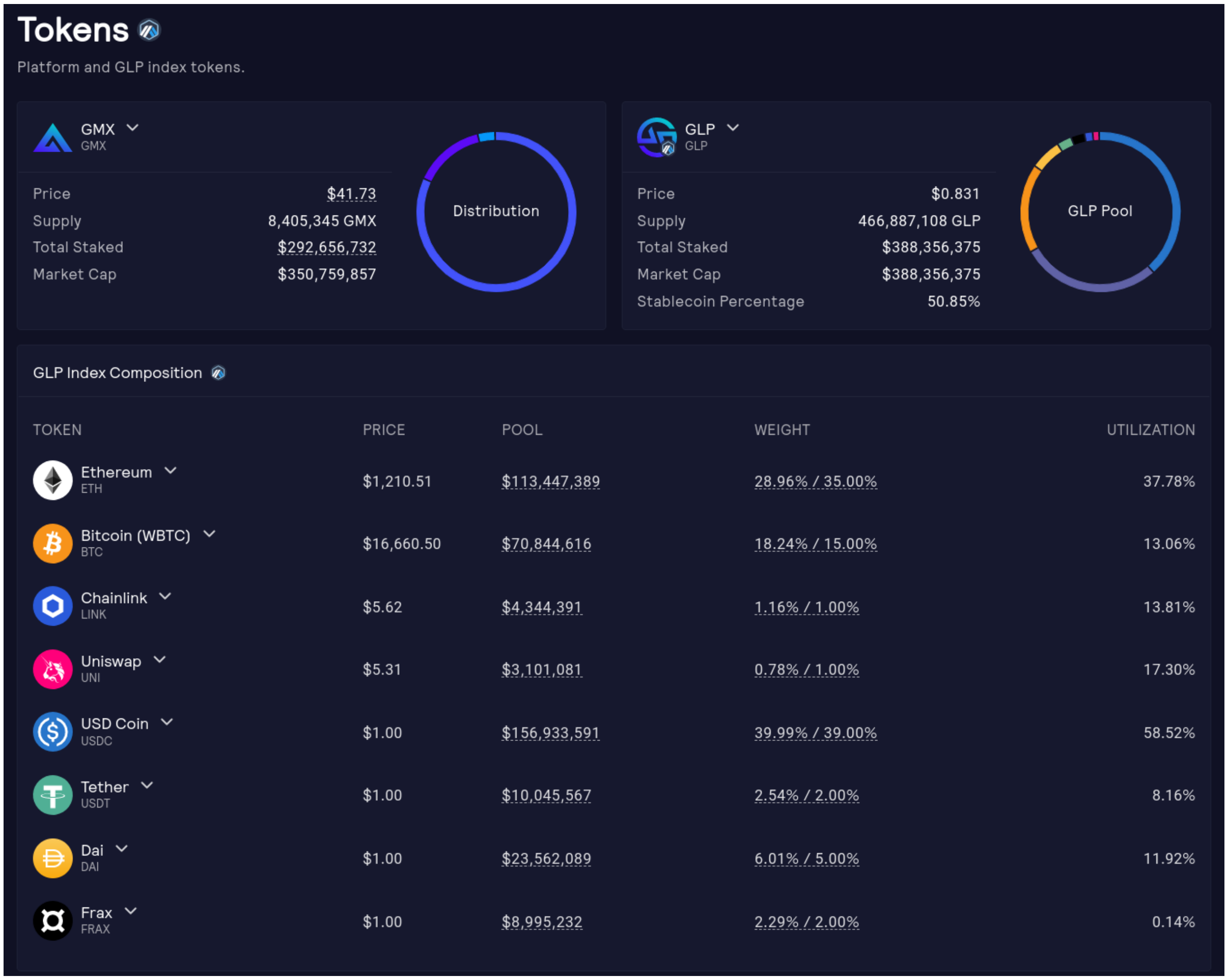
4. Low - Biased delta neutral position

The delta neutral position taken by the junior vault has some imperfections in it. While these are not likely to cause major problems during normal market conditions, the risks of an unbalanced position during certain market edge cases should be considered.

Technical Details

The junior vault attempts to hold equal long and short positions to maintain a delta neutral position in the market. The reason this is done is to avoid impermanent loss, which is a common problem for liquidity providers. The junior vault holds a short position by borrowing WETH and WBTC from Aave (after depositing USDC to maintain a reasonable health factor). The junior vault holds an equal long position by holding GLP (technically fsGLP, the staked version of GLP) which consists of a basket of underlying assets mostly consistently of WETH, WBTC and stablecoins. There are two small ways in which the Rage Trade delta neutral strategy has bias in its holdings that could create unwanted exposure to certain assets. A consistent bias over the long run can impact yield returns to depositors because impermanent loss may cause losses in the long run.

1. The GLP vault does not hold exactly 50% WETH & WBTC with the remaining 50% in stablecoins. It targets a holding of 1% LINK and 1% UNI, so in reality the targeted exposure is 35% WETH, 15% WBTC, 1% UNI, 1% LINK, and 50% stablecoins. The delta neutral strategy has no short position to cover the long exposure held in UNI or LINK. Although the percentage of UNI and LINK holdings is small, this exposure can result in non-zero impermanent loss over the long run. If there is an assumption that UNI and LINK prices are correlated to WETH and WBTC, then the value of the short positions should be increased to roughly 52% of the GLP value, rather than the currently targeted 50%. LINK is an asset that can be borrowed and shorted on Aave like WETH and WBTC currently are, an approach which would maintain a more precise neutral position.



Tokens

Platform and GLP index tokens.

GMX
GMX

Price	$41.73
Supply	8,405,345 GMX
Total Staked	$292,656,732
Market Cap	$350,759,857

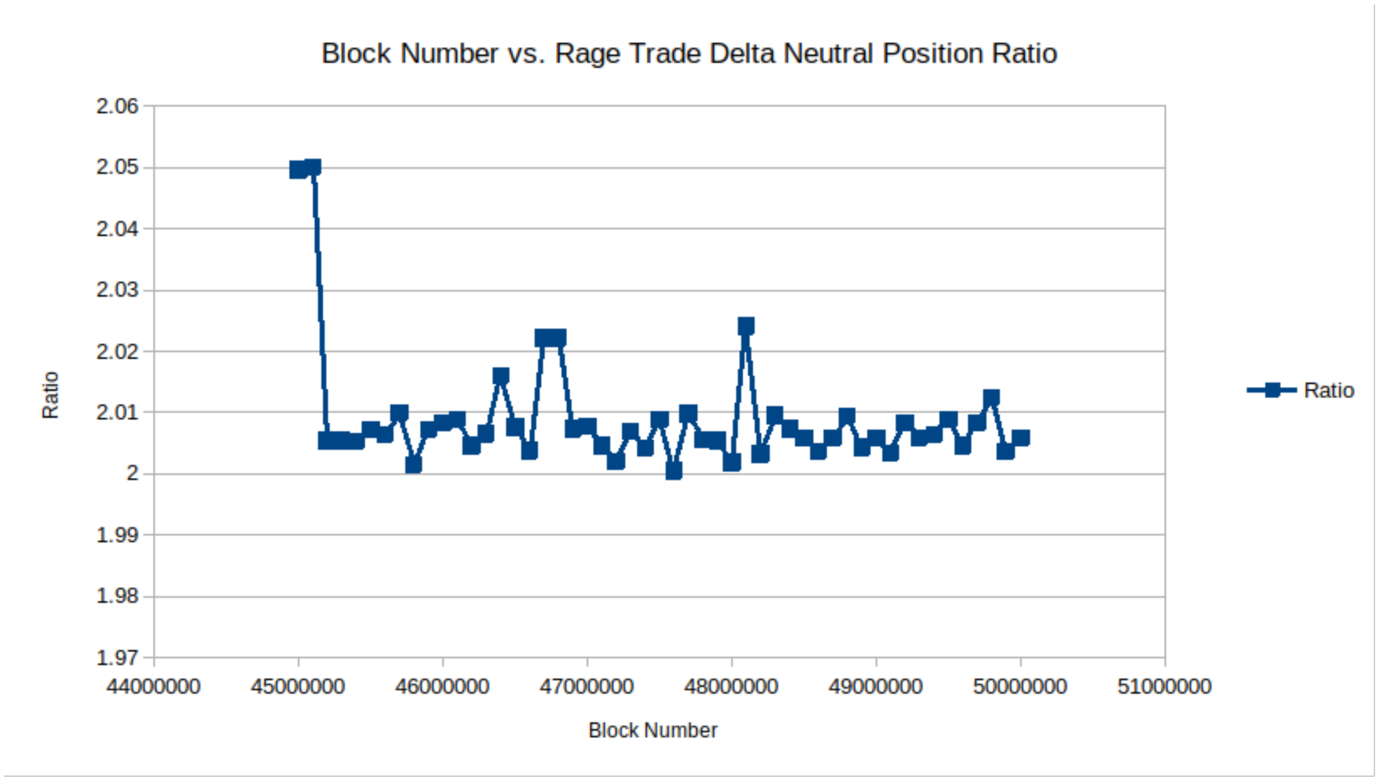
GLP
GLP

Price	$0.831
Supply	466,887,108 GLP
Total Staked	$388,356,375
Market Cap	$388,356,375
Stablecoin Percentage	50.85%

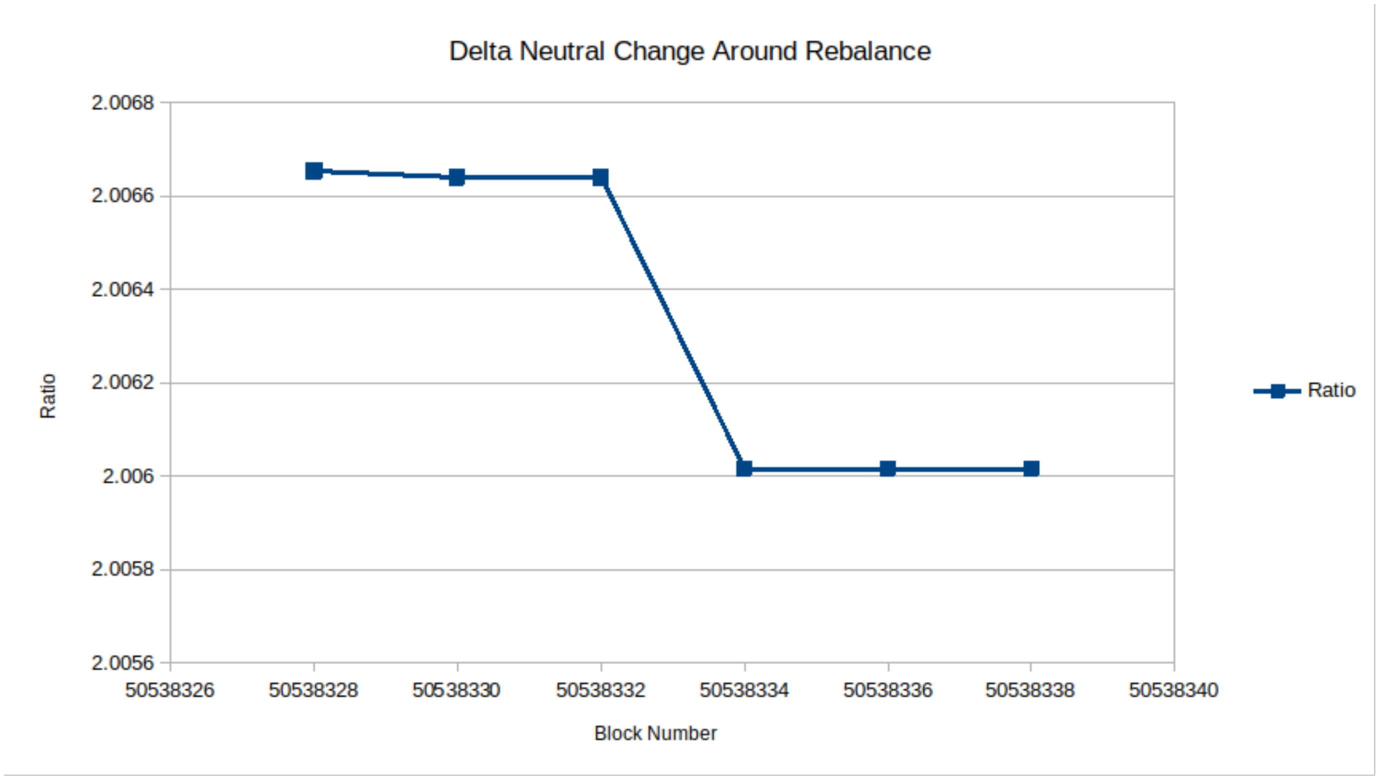
GLP Index Composition

TOKEN	PRICE	POOL	WEIGHT	UTILIZATION
Ethereum ETH	$1,210.51	$113,447,389	28.96% / 35.00%	37.78%
Bitcoin (WBTC) BTC	$16,660.50	$70,844,616	18.24% / 15.00%	13.06%
Chainlink LINK	$5.62	$4,344,391	1.16% / 1.00%	13.81%
Uniswap UNI	$5.31	$3,101,081	0.78% / 1.00%	17.30%
USD Coin USDC	$1.00	$156,933,591	39.99% / 39.00%	58.52%
Tether USDT	$1.00	$10,045,567	2.54% / 2.00%	8.16%
Dai DAI	$1.00	$23,562,089	6.01% / 5.00%	11.92%
Frax FRAX	$1.00	$8,995,232	2.29% / 2.00%	0.14%

1 The second bias is that the process that rebalances the junior vault holdings consistently has more long exposure than short. This can be seen on the long timescale where the ratio is targeting 2 (the reason the target ratio is 2 and not 1 is because only 50% of the holdings in the GMX vault are WETH and WBTC) but the actual value is consistently above 2.



Zooming in on the shorter timescale, observe the point in time when a rebalance happens in the middle of this plot. While the rebalance brings the ratio closer to 2, it could do better, and this inaccuracy may be due to an inaccuracy in the implementation's math.



Impact

Low. Biases built into the code results in small but continuous exposure to assets and impermanent loss.

Recommendation

Consider adjustments to the algorithm that allocates long and short positions to account for and attempt to fix these biases.

Developer Response

Acknowledged, not fixed. LINK and UNI exposure in GMX is very small, which is supposed to decrease even further over time.

Gas Savings Findings

1. Gas - Make functions external

If a function is not called internally, it can be external instead of public for gas efficiency.

Technical Details

Make `getVaultMarketValue()` in the senior vault and the junior vault external, not public. The same can be done for `getPriceX128()` in the senior vault and the junior vault. `getPriceX128()` in the senior vault includes a comment that it is only used in the frontend.

Impact

Gas savings.

Recommendation

Use external functions when possible.

2. Gas - Immutable variables are cheaper

If a variable is not changed multiple times, it can be immutable to save gas.

Technical Details

`pool` can be immutable for gas savings.

Impact

Gas savings.

Recommendation

Make `pool` immutable.

3. Gas - Remove unused function

The internal `getMaxVariableBorrowRate()` is never used and can be remove to save gas on deployment.

Technical Details

`getMaxVariableBorrowRate()` is not used in any contracts in the Rage Trade repo and can be removed.

Impact

Gas savings.

Recommendation

Remove `getMaxVariableBorrowRate()`.

4. Gas - Remove duplicate length check

`_executeFlashloan()` unnecessary verifies that the two input arrays have the same length, using extra gas on every flashloan.

Technical Details

The array length check in `_executeFlashloan()` is unnecessary because it is duplicated a few lines later when the Balancer vault `flashloan()` call happens.

Impact

Gas savings.

Recommendation

Remove the array length check from `_executeFlashloan()`.

Informational Findings

1. Informational - Unnecessary addition operation

There is no need to add a variable to zero, the zero value can be replaced.

Technical Details

`usdcBorrowed` is zero at the start of `totalUsdcBorrowed()`, so the line can be changed to:

```
- if (address(leveragePool) != address(0)) usdcBorrowed +=
leveragePool.getUsdcBorrowed();
+ if (address(leveragePool) != address(0)) usdcBorrowed =
leveragePool.getUsdcBorrowed();
```

Impact

Informational.

Recommendation

Remove unnecessary addition.

2. Informational - No way to remove approvals

The vaults will exist remain behind proxy contracts. If the logic contract is updated, the same approvals with the same tokens will exist from the previous version of the logic contract. It may be useful to have a way to remove these.

Technical Details

`grantAllowances()` in the junior vault and senior vault exists solely to set the allowances for the vault, even though these are set in the `initializer()`. It may also be useful to have a way to remove these allowances. For example, if the Aave pool address is updated or there is a security concern with an approved contract, it may be useful to remove the approval. But there is currently no ability to do this with the vaults. This ability may be useful to introduce because a paused vault will not prevent tokens from being withdrawn by an approved address when an infinite allowance is set.

Impact

Informational.

Recommendation

Consider implementing a `removeAllowances()` function that reverses the actions of `grantAllowances()` for contract upgrades and certain situations where the security of integrated protocols are impacted. This could provide an additional layer of security beyond pausing the vault.

3. Informational - Move Aave pool and oracle update to new function

`setHedgeParams()` has two parts that should probably be separated into two separate functions.

Technical Details

The second half of `setHedgeParams()` is unrelated to the first half and may not be necessary to use each time the other state variables are updated. Consider moving the second half of `setHedgeParams()` to a separate function.

Impact

Informational.

Recommendation

Move the update of `state.pool` and `state.oracle` to a new function because it is unrelated to the rest of `setHedgeParams()`.

4. Informational - Unused dependencies imported

WadRayMath and IRewardTracker are not used in DnGmxJuniorVault.

Technical Details

The WadRayMath import from Aave V3 is never used by DnGmxJuniorVault. The same applies to IRewardTracker.

Impact

Informational.

Recommendation

Remove WadRayMath import from DnGmxJuniorVault.

5. Informational - Hypothetical revert condition breaks EIP4626 requirements

EIP4626 states `totalAssets()` must not revert, but there is a path where it could revert.

Technical Details

EIP4626 states `totalAssets()` has the requirement that it `MUST NOT revert`. But in the senior vault, the call flow `totalAssets()` -> `totalUsdcBorrowed()` -> `dnGmxJuniorVault.getUsdcBorrowed()` -> `.toInt256()` calls a SafeCast function that could revert if the uint256 value in `getUsdcBorrowed()` is greater than or equal to 2**255. It is unlikely that Rage Trade will hold this much value, which would equate to 2**249 dollars of value, but `totalAssets()` would revert in such a case.

Impact

Informational.

Recommendation

Modify `setAdminParams()` to require the `depositCap` not exceed a value that could cause `totalAssets()` to revert. Alternatively, modify SafeCast to remove the revert condition from the `totalAssets()` call flow.

6. Informational - Replace magic numbers with constants

Constant variables should be used in place of magic numbers to prevent typos. For one example, the magic number 10000 is found in multiple places in Swapper.sol and should be replaced with a constant. Using a constant also adds a description to the value to explain the purpose of the value. This will not change gas consumption.

Technical Details

There are many instances of magic numbers with powers of ten. Consider replacing these magic numbers with a constant internal variable, which is already done in some cases like `PRICE_PRECISION`. Some examples are these magic numbers 1e12, 1e14, 1e16, and 1e30.

Impact

Informational.

Recommendation

Use constant variables instead of magic numbers.

7. Informational - Unnecessary reimplementation of GMX functions

The junior vault and junior vault manager duplicate effort by implementing existing code.

Technical Details

The junior vault's `getPrice()` and the junior vault manager's `_getGlpPrice()` functions are identical other than the different of the state function argument. Furthermore, these functions reimplement a function named `getPrice()` in GlpManager that returns the GLP price. In fact, the return value and decimals of the junior vault's `getPrice()` and `getPrice()` in GlpManager is identical, so arguably the function isn't needed at all and the GlpManager contract can be called directly when needed. The same applies to the first half of `_getGlpPriceInUsdc()` in the junior vault manager.

Impact

Informational.

Recommendation

Instead of reimplementing the function from GlpManager, call `state.glpManager.getPrice(maximize)` and adjust the decimals precision as needed, such as dividing by 1e12 to mimic the junior vault `getPrice()` implementation.

8. Informational - Library functions with unusual visibility

Functions that are declared external in a library are not callable outside of the contract that imports the library like a normal external function declared in a contract. Public and external functions in a library are effectively internal functions in the contract they are imported into.

Technical Details

A library is slightly different than a contract in that the functions in a library are only accessible to the contract that imports the library. This means the junior vault manager library can remove any external functions that are declared external in the library but are not called by the junior vault (which is the only contract that imports the library). Such functions include `getTokenReservesInGlp()`, `rebalanceBorrow()`, `getTokenPrice()`, `getGlpPrice()`, `getGlpPriceInUsdc()`, `getTokenPriceInUsdc()`, `getLiquidationThreshold()`, `flashloanAmounts()`, `getOptimalCappedBorrows()`, `getTokenReservesInGlp()`, and `isWithinAllowedDelta()`. Another change that could be made with the library is combining external and internal functions if the external function only calls the internal function.

Impact

Informational.

Recommendation

Remove the external functions in the library that only call internal functions and therefore serve no purpose.

9. Informational - Functions altering state variables missing events

Some functions that modify state variables do not emit events. Events provide a log that can simplify the process of monitor for certain events.

Technical Details

Some functions that modify state variables do not emit events:

- `updateFeeStrategyParams()`
- `unstakeAndVestEsGmx()`

- `stopVestAndStakeEsGmx()`
- `claimVestedGmx()`

Impact

Informational.

Recommendation

Add events to functions that modify state variables for a variety of protocol logging and monitoring use cases.

10. Informational - Dead link in `ERC4626Upgradeable.sol` comment

The solmate link in the comment above `ERC4626Upgradeable` redirects and can be replaced.

Technical Details

This link points to a repo that no longer exists.

Impact

Informational.

Recommendation

Replace the link with this active one.

11. Informational - Move `excessUtilizationRate` calculation in `calculateFeeSplit()` into if statement

`excessUtilizationRate` is calculated even though it isn't used in the else case. Moving the calculation into the if statement saves gas in the else case.

Technical Details

This line can be moved inside the if statement.

Impact

Informational.

Recommendation

Move the calculation into the if branch, so it isn't calculated unnecessarily in else cases.

Final remarks

engn33r

The approach to achieving a neutral strategy is quite clever, creative, and combines existing DeFi lego blocks in a unique way. The high level design of the system generally makes sense and offers attractive benefits for depositors in the junior or senior tranche. The code could use refactoring in some placing to better compartmentalize and group certain functionalities. Tracing certain code paths was a complex process, and complexity is not good for system security. Additional modelling of the protocol, ideally using unit tests to rely on the implementation in the code, would assist with improving the robustness of the strategy design in adverse market condition.

Invader-tak

Rage Trade takes advantage of cheap gas costs on Arbitrum to introduce really innovative DeFi interaction across several protocols (GMX, Aave, Balancer) built on top of a tranching protocol. The downside of this is that Rage has introduced a high level of complexity in its attempts to ensure stable, high yields, and protection for the senior tranche. This complexity means that there are a lot of possible attack vectors that have to be explored in order to establish protocol correctness. I've spent a lot of time fuzzing the code to see if there are any scenarios that would allow the protocol to be exploited, but would recommend that more time is spent exploring protocol invariants and building fuzzing campaigns to determine their correctness.