yAudit Rage Trade Upgrade Review

Review Resources:

- Rage Trade Docs

Auditors:

- engn33r
- Invader-tak

Table of Contents

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 2 auditors between February 15, 2023 and March 3, 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 2019b32085cc9df369b82e27520af437ef0ba914 for the Rage Trade repo. Additional fixes were added during the review so some findings are made referencing the new commit hash 793c48e86db99e75561f0619c3030f7f89f81f66.

Scope

The scope of the review consisted of all the contracts in the repo at the specific commit. The goal was the review the changes made since the previous yAudit review. Most of the contract changes were made in PR #71 and PR #84. 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.

yAudit 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

Category	Mark	Description
		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. However, such math and complexity is not easily avoided for a delta neutral strategy.
Complexity	Average	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. However, some of the logic around rebalancing has been improved and clarified since the first review so it is a little easier to understand. If paired with thorough user documentation, it should be even more approachable for developers to integrate with Rage Trade.
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. The complexity added from 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 changes. However, this is expected for a protocol that is not aiming for an immutable deployment.
Code stability	Average	Rage Trade has launched on mainnet and should be ready for an upgrade after this review. However, partway through the audit an important PR was added to the scope of the review.
Documentation	Average	NatSpec is absent from some functions as noted in an info finding. The official docs could add some details that would be useful for anyone examining the Rage Trade

Category	Mark	Description
		design or implementation, such as elaborating on how the 2.3x leverage number for the senior vault is calculated.
Monitoring	Good	Key functions that modify state variables emit events, including the functions that did not have events during the initial Rage Trade review.
Testing and verification	Average	Some issues were encountered running the tests at this specific commit, but given there are more test cases since the last review it is expected that the tests maintain good coverage of the code in the protocol.

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

None.

Medium Findings

1. Medium - Inaccurate rounding from `maximize`

The `maximize` value is incorrect in some locations in DnGmxJuniorVaultManager.sol.

Technical Details

There are two locations where `mulDivDown()` uses `_getGlpPriceInUsdc(state, maximize)` in the denominator and one place where `_getGlpPriceInUsdc(state, !maximize)` is in the denominator. This inconsistency indicates a bug.

If a fraction is intended to be maximized (with `maximize` set to true), then the denominator should be reduced (with `maximize` set to false). If the fraction is intended to be minimized (with `maximize` set to false), then the denominator should be increased (with `maximize` set to true). This indicates that the correct value of `maximize` when it is used in the denominator is inverted from the value it holds in the current function. The two cases where this is not done, line 1140 and line 1146, should be fixed to align to line 1172.

Impact

Medium. Inaccuracy in the value can result due to the wrong maximize value.

Recommendation

Invert the `maximize` boolean value when it is used in `_getGlpPriceInUsdc()` in the denominator of `mulDivDown()`.

Developer Response

Acknowledged. Will change `maximize` to `!maximize` only for line 1140 and not for line 1146. Since to maximize `totalAssets`, `aaveProfits` should be maximized (hence denominator should be minimized) whereas `aaveLoss` should be minimized (hence denominator should be maximized) Fix Commit

2. Medium - Loss of precision can lead to loss of value

`rewardRouter.unstakeAndRedeemGlp()` takes a _minOut argument to limit slippage. This value is rounded down in `_convertToToken()` of WithdrawPeriphery.

Technical Details

The calculation of `minTokenOut` in `_convertToToken()` is done in a way where the value is reduced to 6 decimals of precision, then increased to the number of decimals needed for

that specific token. This process means that all digits beyond the first 6 decimals of precision will be zeros. This effectively rounds down the value, which increases the amount of possible slippage from the value targeted by the `slippageThreshold` value. This could result in minor loss of value.

Impact

Medium. Weak slippage settings due to minTokenOut rounding down can result in a loss of value.

Recommendation

Replace the existing code with a single line to avoid loss of precision.

```
- uint256 minTokenOut = outputGlp.mulDiv(glpPrice * (MAX_BPS - slippageThreshold), tokenPrice * MAX_BPS);
- minTokenOut = minTokenOut * 10 ** (IERC20Metadata(token).decimals() - 6);
+ uint256 minTokenOut = outputGlp.mulDiv(glpPrice * (MAX_BPS - slippageThreshold) * (10 ** (IERC20Metadata(token).decimals() - 6)), tokenPrice * MAX_BPS);
```

Developer Response

Acknowledged, will fix according to the recommendation Fix Commit

Low Findings

1. Low - Possible underflow could cause revert

There is a subtraction operation that may underflow and cause a revert.

Technical Details

A comment in `_executeVaultUserBatchStake()` attempts to explain why `_roundUsdcBalance` will be always >= `_usdcToConvert`, but the logic does not hold up. Take the following example scenario.

Starting values:

- `usdcAmountToConvert` = 1
- `_roundUsdcBalance` = 2
- `minUsdcConversionAmount` = 4

Logical steps

1. After line 388: `_usdcToConvert` = 1
2. After line 397: `_usdcToConvert` = 4
3. `_roundUsdcBalance - _usdcToConvert` will underflow on line 399 because `_roundUsdcBalance` = 2 while `_usdcToConvert` = 4

The same logic is found in DnGmxBatchingManagerGlp.sol.

Impact

Low. A revert is possible, but the function is protected with the `onlyKeeper` modifier so it may not have a direct impact on users.

Recommendation

If this edge case could cause problems, modify the comments and logic to address this edge case where `_roundUsdcBalance` < `minUsdcConversionAmount`. Consider using the equivalent check without subtraction, which is `_roundUsdcBalance <= minUsdcConversionAmount + _usdcToConvert`.

Developer Response

Not an issue. There is an additional check in deposit function on line 210 which ensures that a deposit < `minUsdcConversionAmount` cannot occur. This ensures that `_roundUsdcBalance` >= `minUsdcConversionAmount`. Hence this case can never occur.

2. Low - Cannot undo infinite approval

An infinite approval of any address is generally done to save gas, but it can be useful to have a failsafe to remove such approvals.

Technical Details

`grantAllowances()` in the batching managers gives an infinite approval of sGLP to dnGmxJuniorVault, but there is no way to undo this approval. It may be useful to add a function argument to `grantAllowances()` to allow a custom approval value to be set, including an approval of zero. While the batching managers are implemented behind proxies, in the event that the approval needs to be removed, response time will likely be important so it is preferable to have failsafes in place before they are needed. Because dnGmxJuniorVault is implemented behind a proxy, the code at that address today may differ from the code at that address in the future, making safety measures more important.

Impact

Low. Infinite approval cannot be undone without an upgrade.

Recommendation

Modify `grantAllowances()` to take a function argument that allows the owner to set the approval allowance.

Developer Response

Acknowledged, not fixing. The approvals are given to either rage-trade's own deployed contract or gmx/uniswap contracts which are immutable and cannot upgrade logic to misuse it.

3. Low - `abs()` reverts for `type(int256).min`

The implementation of `abs()` does not use unchecked and reverts for `type(int256).min`.

Technical Details

The `abs()` implementation in SignedFixedPointMathLib is not from OpenZeppelin's SignedMath. OpenZeppelin's code has a comment indicating unchecked must be used to support the argument value of `type(int256).min`. The current `abs()` implementation reverts on this value. Because `mulDivDown()` relies on this `abs()` implementation, it too would revert in the case of `x == type(int256).min`.

Impact

Low. Math functions should not revert for any input value in valid range.

Recommendation

Use OpenZeppelin's implementation of `abs()`.

Developer Response

Acknowledged, will fix it as per the recommendation. Fix Commit

4. Low - Missing assert risks casting overflow

Safemath does not apply to casting, to a casting operation may overflow. There is a missing check to prevent against this overflow case.

Technical Details

`_calculateSwapLoss()` contains two casting operations where a value is negated. In the second case, it is clear that the negation will make `tokenAmount` positive because

`tokenAmount` is less than zero in that code branch. But in the first case, it is not clear that the negation will make `otherTokenAmount` positive rather than negative. `_calculateSwapLoss()` is called with values returned from `QuoterLib.quoteCombinedSwap()`. `quoteCombinedSwap()` contains an assert that validates `ethAmountInEthSwap` and `usdcAmountInEthSwap` have opposite signs. But there is no similar check to confirm `btcAmountInBtcSwap` and `usdcAmountInBtcSwap` have opposite signs. This means the `_calculateSwapLoss()` call with BTC values may have a chance of a casting overflow because there is no assertion to guarantee avoidance of this case.

Impact

Low. Potential risk of edge case casting overflow.

Recommendation

Add assert in `QuoterLib.quoteCombinedSwap()` for BTC values like is done with ETH values.

Developer Response

Acknowledged. The QuoterLib._getQuote (line 131), there are 3 cases and first two cases take an unsigned integer and apply sign over it such that one of them is positive and other is negative. Third case is just both of them zero. Hence btcAmountInBtcSwap and usdcAmountInBtcSwap would have opposite signs. However to make the code more readable as well as prevent any future issues, we would add the assert statement. Fix Commit

Gas Savings Findings

1. Gas - Combine zero checks

Two zero checks can be combined.

Technical Details

In batching manager, when `executeBatch()` is called it checks if the function argument is zero and then later checks if a value derived from the argument is zero. The first check can be removed, because if the function argument is zero it will be caught in the second check. The same logic is in DnGmxBatchingManagerGlp.sol.

Impact

Gas savings.

Recommendation

Consider this gas savings in the two batching manager contracts.

2. Gas - Use unchecked if no underflow risk

There is a subtraction operation that can use unchecked for gas savings.

Technical Details

There is at least one example where unchecked can be applied, because the line before the subtraction prevents an underflow:

```
- userDeposit.unclaimedShares = userUnclaimedShares - amount.toUint128();
+ unchecked { userDeposit.unclaimedShares = userUnclaimedShares - amount.toUint128();
}
```

The same modification could be made to DnGmxBatchingManagerGlp.sol.

Impact

Gas savings.

Recommendation

Use unchecked block if there is no overflow or underflow risk for gas savings. Consider adding a fuzzing test to verify the underflow cannot happen to add a safety check for future code modifications.

3. Gas - Use cached value

`userDeposit.usdcBalance` is a storage variable that is already cached. The cached value can be used to save gas.

Technical Details

`userDeposit.usdcBalance` is cached in `userUsdcBalance` in `depositUsdc()`. This cached value can be used instead of `userDeposit.usdcBalance` on line 225. This optimization is already used later in the same contract. The same edit can be made in DnGmxBatchingManagerGlp.

Impact

Gas savings.

Recommendation

Use cached variables when available.

4. Gas - Remove unused functions

`simulateSwap()` in Simulate.sol could be deleted. The same applies to the `onlyDnGmxJuniorVault` modifier.

Technical Details

`simulateSwap()` with 4 function arguments is not used anywhere and can safely be removed from Simulate.sol to save some gas on deployment.

The `onlyDnGmxJuniorVault` modifier is declared in DnGmxBatchingManager and DnGmxBatchingManagerGlp but is never used in Rage Trade. These modifier declarations can be removed.

Impact

Gas savings.

Recommendation

Remove unused functions.

Informational Findings

1. Informational - Consider refactoring

Two nearly identical functions could be combined.

Technical Details

`_getMaxTokenHedgeAmount()` is nearly identical to `_getTokenHedgeAmount()`. The only difference is in the last line of these functions. `_getTokenHedgeAmount()` could be replaced with `_getMaxTokenHedgeAmount() * _traderOIHedgeBps / MAX_BPS`.

Impact

Informational.

Recommendation

Consider combining similar functions.

2. Informational - Update and improve comments

Some functions need to have their comments updated after recent changes. Elsewhere, two similar functions have identical comments even though they return slightly different values.

Technical Details

Several improvements can be made in comments:

1. DnGmxBatchingManagerGlp.sol now handles USDC instead of GLP. Many comments still reference GLP but should now mention USDC. For example, this comment and this comment in `depositUsdc()` should say USDC not sGLP. There are other comments that also need updating to reflect the switch to using USDC.
2. `slippageThresholdGmxBps` in DnGmxBatchingManager is used for USDC slippage but also WETH slippage in `rescueFees()`. This comment only mentions USDC but should mention WETH.
3. `_getMaxTokenHedgeAmount()` is nearly identical to `_getTokenHedgeAmount()` but the comments of the two functions are identical. The difference is that `_getTokenHedgeAmount()` returns a value multiplied by `_traderOIHedgeBps`. This difference in not reflect in the comments of the function.

Impact

Informational.

Recommendation

Update comments as the code changes and differentiate comments of similar functions.

3. Informational - Efficiency improvement

A pure function performs unnecessary operations.

Technical Details

The `tokenTraderOIMax` function argument for the function `_checkTokenHedgeAmount()` is returned from `_getMaxTokenHedgeAmount()`. This returned value should always be positive. This means `tokenTraderOIMax.sign() == 1`, so `_checkTokenHedgeAmount()` can be simplified with the following:

```
function _checkTokenHedgeAmount(int256 tokenTraderOIHedge, int256 tokenTraderOIMax)
internal pure returns (bool) {
```

```
-         if (tokenTraderOIHedge.sign() * tokenTraderOIMax.sign() < 0) return false;
+         if (tokenTraderOIHedge.sign() < 0) return false;
-         if (tokenTraderOIHedge.abs() > tokenTraderOIMax.abs()) return false;
+         if (tokenTraderOIHedge > tokenTraderOIMax) return false;

          return true;
}
```

Impact

Informational.

Recommendation

Consider making the above efficiency improvements for `_checkTokenHedgeAmount()`.

4. Informational - Migration requires managing slot reuse

DnGmxBatchingManager.sol is repurposing a storage slot for a new value with different units. Handling this storage slot value must be done properly during the upgrade process.

Technical Details

In DnGmxBatchingManager.sol, the `glpDepositPendingThreshold` variable in the old contract version is renamed to `minUsdcConversionAmount` in the new contract version. This storage slot used to store an amount of GLP but now stores an amount of USDC. If the logic contract behind the proxy is upgraded before this storage slot value is changed, a well-timed attack may be able to take advantage of the incorrect value.

Similarly, the deprecated variables should be set to zero before the upgrade. While `roundGlpDepositPending` is already set to zero, the private `bypass` variable is not, so `setBypass()` should be called before the upgrade, otherwise the variable will remain set. `batchingManager` in

Impact

Informational.

Recommendation

Make sure the contract upgrade behind the proxy and the updating of key storage slots happen in the same transaction.

5. Informational - Use updated solmate import

solmate is imported from Rari Capital, but the maintained repository is at transmissions11/solmate.

Technical Details

Consider replacing the outdated reference in package.json and multiple solidity files to use the current location of the solmate library. This will help to ensure the latest bug fixes and improvements are used.

Impact

Informational.

Recommendation

Upgrade the solmate imports.

6. Informational - Possible mulDivDown confusion

There are two `mulDivDown()` implementations used in DnGmxJuniorVaultManager.sol with the same name and same number of function arguments. This could cause confusion over which one is used if there is an implementation difference between them.

Technical Details

Consider renaming the custom `mulDivDown()` implementation to indicate the first argument is a int256, not a uint256 like the solmate implementation of `mulDivDown()`. One place where this ambiguity may cause confusion is this line. Solidity will prevent an underflow from happening, so the solmate implementation will be used if no revert happens, but should an underflow be allowed here with unchecked to allow the custom implementation to be used?

Impact

Informational.

Recommendation

Consider renaming the custom function to `mulDivDownInt()` or other modifications to avoid two very similar function names.

7. Informational - Unnecessary operations

There is an unnecessary casting and an unnecessary constant value in an event.

Technical Details

This line unnecessarily casts `netSlippage` to uint256 when it is already of that type.

This line emits an event with the constant value of `address(0)`. Because this event is used in only this one location, there is no point in this event taking a constant value.

Impact

Informational.

Recommendation

Remove unnecessary casting.

8. Informational - Change variable name

The same variable name is used in two places to refer to different values. Consider changing one variable name.

Technical Details

The variables name `unhedgedGlp` is used in two places in DnGmxJuniorVaultManager. Only the second case of this variable actually refers to the exact value of unhedged GLP maintained by Rage Trade, while the first case actually refers to the target amount of unhedged GLP that the protocol should rebalance to.

Impact

Informational.

Recommendation

Consider renaming this instance of `unhedgedGlp` to `targetUnhedgedGlp`.

9. Informational - Missing NatSpec comments

Some variables and functions are missing NatSpec comments to explain their purpose. Because the Rage Trade docs are high levels docs, developers working on integrations will likely rely on NatSpec to better understand the protocol.

Technical Details

Several variables and functions could benefit from improved NatSpec:

1. `depositCap`: add the comment “Maximum USDC that can be deposited in one round”
2. `depositCap`: add the comment “Maximum sGLP that can be deposited in one round”

3. `depositUsdc()`, `executeBatch()`, `claimAndRedeem()`, `rescueFees()`, and all internal functions are missing NatSpec in DnGmxBatchingManager.sol.
4. `batchingManager` variable in State struct should be marked as deprecated because batching manager integration was removed.
5. `_getBorrowValue()` should have a comment to specify the return value is in USDC not USD

Impact

Informational.

Recommendation

Add NatSpec where missing.

10. Informational - Typos in comments

Some typos exist in comments.

Technical Details

These typos were found in comments:

- cooldowm -> cooldown
- depoists -> deposits
- roudn -> round
- accumuated -> accumulated
- mangager -> manager
- cooldowm -> cooldown
- depoists -> deposits
- roudn -> round
- accumuated -> accumulated
- flase -> false (found here, here, here, and here)
- glp price in usd -> glp price in usdc (found here and here)
- againts -> against
- assests -> assets
- retunrs -> returns

Impact

Informational.

Recommendation

Fix typos.

Final remarks

The mitigations reviewed during this audit demonstrated that the findings from the previous review were understood and fixed. Overall the design shows a lot of thought around key elements of the protocol, and the ability to upgrade contracts and change key parameters is useful for managing positions in a complex strategy.

One item that was not thoroughly examined during testing but is crucial for the protocol is thorough testing of the upgrade process. A fork of mainnet should undergo the upgrade process and all the key values, such as totalAssets and other values, should be queried to validate they match the expected values, or are the values as the current mainnet deployment. The setup process to test the upgrade deployment against a mainnet fork was not easy to perform at the time the testing happened.