Security Review
NM-0051: Gyroscope

NETHERMIND

Aug 17, 2022

Contents

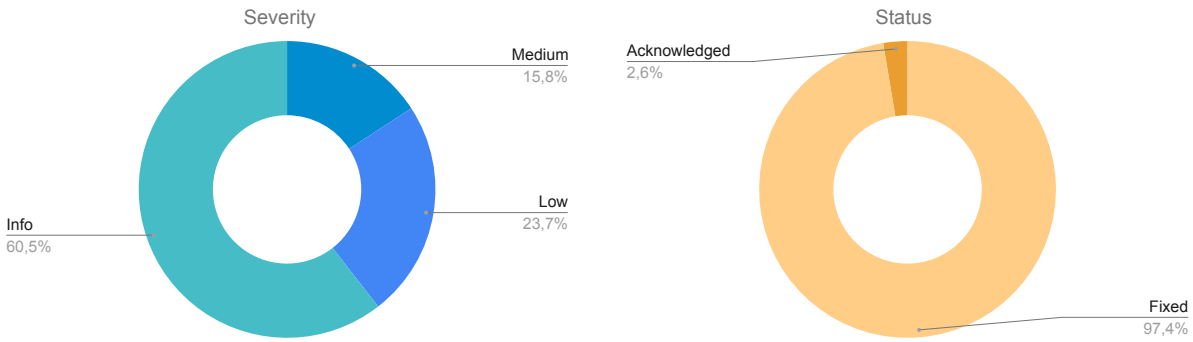
1 Executive Summary

This document presents the results of the security audit performed by Nethermind on the source code for the Gyroscope protocol. Gyroscope is a fully-backed stablecoin with autonomous price bounding and all-weather reserves.

During this audit two repositories were reviewed: protocol and vaults. The vaults repository contains the implementation for three types of Balancer pools that are used as secondary markets for the protocol. The protocol repository contains the implementation of the core components of the protocol, including the Primary Market where Gyroscope token will be minted and redeemed.

The audit consisted of a manual inspection of the code base. Knowledge of the protocol was obtained from papers provided by the Gyroscope team, multiple walkthroughs from the client and public documentation of the protocol.

This document reports multiple findings uncovered during the review process. A total of 38 findings are reported, no critical issue was found. Most of the findings reported are Informational, these are suggestions that may improve code quality. Some of the main issues reported are related to the use of *orders* in the **Motherboard** contract, these *orders* contain all of the vaults over and above the assets included in the operation's arguments. Other primary issues are due to differences between the PAMM paper and the implementation. In general, the Gyroscope codebase is well written and has an extensive test suite.



(a) (b)

Critical (0), **High** (0), **Medium** (6), **Low** (9), **Undetermined** (0), **Informational** (23), **Best Practices** (9).

Summary of the Audit

Audit Type	Security Review
Initial Report	May 18, 2022
Response from Client	Aug 10, 2022
Final Report	Aug 17, 2022
Methods	Manual Review, Automated Analysis
Repository	Vaults, Protocol
Commit Hash Vaults	79f42be9388d4834cee085f327360f5ead2a66ec
Commit Hash Protocol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
Documentation	https://docs.gyro.finance/overview/introduction

2 Audited Files

2.1 Vaults Repository

Reviewed Files

File	Git commit hash
contracts/cpmmv-2/Gyro2PoolErrors.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-2/GyroTwoMath.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-2/GyroTwoOracleMath.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-2/GyroTwoPool.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-2/GyroTwoPoolFactory.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-3/GyroThreeMath.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-3/GyroThreePool.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-3/GyroThreePoolErrors.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cpmmv-3/GyroThreePoolFactory.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cemm/GyroCEMMMath.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cemm/GyroCEMMOracleMath.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cemm/GyroCEMMPool.sol	79f42be9388d4834cee085f327360f5ead2a66ec
contracts/cemm/GyroCEMMPoolErrors.sol	79f42be9388d4834cee085f327360f5ead2a66ec
libraries/GyroPoolMath.sol	79f42be9388d4834cee085f327360f5ead2a66ec

2.2 Protocol Repository

Reviewed Files

File	Git commit hash
contracts/auth/Governable.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/exchangers/BalancerExchanger.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/exchangers/BalancerPoolRegistry.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/fee_handlers/StaticPercentageFeeHandler.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/FeeBank.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/GydToken.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/GyroConfig.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/LPTokenExchangerRegistry.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/Motherboard.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/AssetRegistry.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7

Reviewed Files

File	Git commit hash
contracts/oracles/balancer/BalancerCEMMPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/balancer/BalancerCPMMPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/balancer/BalancerCPMMV2PriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/balancer/BalancerCPMMV3PriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/balancer/BalancerLPSharePricing.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/balancer/BaseBalancerPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/BaseChainLinkOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/BaseVaultPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/BatchVaultPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/ChainLinkPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/CheckedPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/CrashProtectedChainLinkPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/TrustedSignerPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/oracles/UniswapV3TwapPriceOracle.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/PrimaryAMMV1.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/Reserve.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/ReserveManager.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/safety/ReserveSafetyManager.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/safety/RootSafetyCheck.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/safety/VaultSafetyMode.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/VaultRegistry.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/vaults/BalancerPoolVault.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
contracts/vaults/BaseVault.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
libraries/DataTypes.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
libraries/Flow.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7
libraries/SignedFixedPoint.sol	03f74bd5b3080abf9c684c59458d310ff5fb64d7

3 Summary of Findings

Summary of Findings

Finding	Severity	Update
Vaults with no fees set would disable minting and redeeming	Medium	Fixed
getPriceUSD computes the wrong value	Medium	Fixed
Price safety for reserves is not correctly verified	Medium	Fixed
Difference between PAMM paper and *createDerivedParams* function	Medium	Fixed
Difference between PAMM paper and *isInThirdRegionHigh* function	Medium	Fixed
Difference between PAMM paper and *computeReserveValueRegion* function	Medium	Fixed
Wrong constant value for **Sqrt(1e-17)**	Low	Fixed
Error code duplicated	Low	Fixed
_calculateCurrentValues is not always called with upscaled balances	Low	Fixed
GyroThreePool.sol allows AssetManagers to be set at construction	Low	Fixed
Governance could be lost if wrong address is used in *changeGovernor*	Low	Fixed
BPT tokens in **BalancerExchanger.sol** could be taken by anyone	Low	Fixed
deregisterPoolId only deregisters first occurrence of the pool	Low	Fixed
Disparity between values used in order and values transferred to the reserve	Low	Fixed
getReserveState can revert without emitting message	Low	Fixed
Duplicated code in **NewtonSqrt.sol** file	Info	Fixed
Non uniform use of term names in comments	Info	Fixed
Division before multiplication	Info	Fixed
Comments are pointing to the wrong section	Info	Fixed
Comments are not aligned with the code	Info	Fixed
Unused functions in **GyroCEEMMath.sol**	Info	Fixed
Typo in comments	Info	Fixed
Comments are not aligned with the code	Info	Fixed
getSqrtParameters can return a tuple of **uint256** to save gas	Info	Fixed
Duplicated comment in **GyroTwoPool.sol**.	Info	Fixed
Reduced readability for *_onSwapGivenIn* and *_onSwapGivenOut*	Info	Fixed
Unused arguments in **GyroTwoPool.sol** constructor	Info	Fixed
Typo in commented equation in **GyroTwoMath.sol**	Info	Fixed
Number of operations in *_calculateCubicTerms* can be reduced	Info	Acknowledged
_calcNewtonDelta has an unused named argument	Info	Fixed
deposit and *withdraw* do not allow minimum expected amounts to be specified	Info	Fixed
Use of *transferFrom* instead of *safeTransferFrom*	Info	Fixed
Incorrect casing for variable name in *registerPoolId* function	Info	Fixed
Unnecessary storage write	Info	Fixed
Defined event is never emitted in **VaultSafetyMode.sol**	Info	Fixed
Motherboard.sol can *mint* tokens directly to *msg.sender* to save gas	Info	Fixed
setInitialPrice can be called multiple times for the same vault	Info	Fixed
Vault metadata is still available after deregistering the Vault	Info	Fixed

4 Findings

F01: [MEDIUM] Vaults with no fees set would disable minting and redeeming

Context: StaticPercentageFeeHandler.sol#L39, Motherboard.sol#L77, Motherboard.sol#L110

Description:

In the contract **StaticPercentageFeeHandler.sol** the function *applyFees* will revert if at least one of the vaults in *order* does not have its fees set. This function is called from the *mint* and *redeem* functions in the **Motherboard.sol** contract, the *Order* passed as an argument contains all the vaults registered up until the call. If at least one of these vaults does not have registered fees the *mint* or *redeem* call will fail. The process for registering a vault does not include any calls to *setVaultFees*. This increases the possibilities of a vault being registered without any fees being set.

Recommendation:

Use default values for fees when registering a vault.

Status: Fixed.

Update from client: Fixed in commit b06eb9fba50a42a3b49d25ba48d3272779063928

F02: [MEDIUM] *getPriceUSD* computes the wrong value

Context: BaseVaultPriceOracle.sol#12

Description:

The *getPriceUSD* function returns the value of a *vault token* in **USD** using the value in **USD** of the *underlying token*. To compute the price of the *vault token*, the total value of the pool should be computed, this can be done by multiplying the price of the *underlying token* by the amount of *underlying token* in the vault. After computing the total value of the vault, it can be divided by the total amount of *vault tokens* and the result will be the price of *vault tokens* in **USD**. This reasoning can be expressed as $p_u * t_u/t_v$ where p_u is the price of the *underlying token*, t_u is the amount of *underlying tokens* in the vault and t_v is the total amount of *vault tokens*. The exchange rate is defined as *totalUnderlying_* divided by *totalSupply* of the vault, so the value computed by the *getPriceUSD* function can be expressed as $p_u/t_u/t_v$ which is not equivalent to the previous formula.

Recommendations:

Apply the following change on the indicated line

```
- return poolTokenPriceUSD.divDown(vault.exchangeRate());
+ return poolTokenPriceUSD.mulDown(vault.exchangeRate());
```

Status: Fixed.

Update from client: Fixed in commit 385ea8a6082eef2622f23f8cb80e8ef1b0ccb7b8

F03: [MEDIUM] Price safety for reserves is not correctly verified

Context: ReserveSafetyManager.sol#L186

Description:

The comments for the function *_updateVaultWithPriceSafety* state that all prices for non-stable tokens must be bigger than *minTokenPrice*. The variable *vaultMetadata.allTokenPricesLargeEnough* stores a *boolean* indicating if this condition holds. This variable is initialised to *false*, but is updated to *true* every time a token with the correct price is analyzed. If there are multiple tokens of which at least one has a price above the minimum token price, then the final value of *vaultMetadata allTokenPricesLargeEnough* will still be *true*.

Recommendation:

Use the same approach used in *_updateMetaDataWithEpsilonStatus*: initially set the value of *vaultMetadata.allTokenPricesLargeEnough* to *true*, and, if some token has an incorrect price, set it to *false*.

Status: Fixed.

Update from client: Fixed in commits a504026302737b9c77bc897f91fa46ca72dab365 and e6643a51000513d46c714a5c9b4d2749170fd89e

F04: [MEDIUM] Difference between PAMM paper and *createDerivedParams* function

Context: PrimaryAMMV1.sol#L228

Description:

In the PAMM paper, algorithm 1 describes how *DerivedParams* is computed. This algorithm uses *alphaThresholdIIIHL* as the *alpha* argument, but the code in *createDerivedParams* uses *params.alpha*.

Recommendation:

Review inconsistencies between the paper and the implementation and fix as appropriate.

Status: Fixed.

Update from client: Fixed in commit 71ae18c366b7a455fd4d7521ea612abae592b8b7

F05: [MEDIUM] Difference between PAMM paper and *isInThirdRegionHigh* function

Context: PrimaryAMMV1.sol#296

Description:

In the PAMM paper, algorithm 2 describes how to detect the region. To check if the actual region is *Third Region High*, the algorithm computes the reserves using *xlThresholdIIIHL*, but in the implementation *xlThresholdIIHL* is used.

Recommendation:

Review inconsistencies between paper and implementation and fix as appropriate.

Status: Fixed.

Update from client: Fixed in commit 558739d57bf5d290bf8fa388d25946d5eda097ca

F06: [MEDIUM] Differences between PAMM paper and *computeReserveValueRegion* function

Context: PrimaryAMMV1.sol#321

Description:

In the PAMM paper, algorithm 2 describes how to detect the region. For detecting case I) ii), the condition from the paper is $b/y \leq 1 - \bar{\alpha}(x - \bar{x}_U)$, where b corresponds to the program variable $reserveValue$, y to $totalGyroSupply$, $\bar{\alpha}$ to $alphaBar$, x to $redemptionLvel$ and $\bar{x}_U$ to $xuBar$. However, the implementation uses $redemptionLevel \leq xlThresholdAtThresholdI$.

Recommendation:

Review inconsistencies between the paper and the implementation and fix as appropriate.

Status: Fixed.

Update from client: Fixed in commit fcdaa3a112ecbcef6c5c05d8a57de7e3a019f4f2

F07: [LOW] Wrong constant value for Sqrt(1e-17)

Context: GyroPoolMath.sol#L32

Description:

SQRT_1E_NEG_17 is missing a 0, $sqrt(1e-17) = 3.16...e-09$ but the number indicated is $3.16...e-10$

Recommendation:

```
- uint256 private constant SQRT_1E_NEG_17 = 316227766;
+ uint256 private constant SQRT_1E_NEG_17 = 3162277660;
```

Status: Fixed.

Update from client: Constant was incorrect. Fixed in commit 0c23f27724abc5b43a9c43ac5c5a9138cd7ff752.

F08: [Low] Error code duplicated

Context: GyroCEMMPoolErrors.sol#L27

Description:

The **DERIVED_ZETA_WRONG** and **STRETCHING_FACTOR_WRONG** errors share the same *Error Code*. This could be misleading for users.

Recommendation: Change the error code representing **DERIVED_ZETA_WRONG** or **STRETCHING_FACTOR_WRONG**.

Status: Fixed.

Update from client: Fixed in commit cff42fff9407223ccb00e2de869c47846d0941c9.

F09: [Low] *_calculateCurrentValues* is not always called with upscaled balances

Context: GyroTwoPool.sol#L77, GyroTwoPool.sol#207

Description:

calculateCurrentValues and *getVirtualParameters* call *_calculateCurrentValues* without upscaling balances. *getVirtualParameters* gets the balances from the vault and does not upscale these balances before calling *_calculateCurrentValues*. *calculateCurrentValues* is a public function and is not called from inside the contract, it should be documented that balances need to be upscaled.

Recommendation:

- Upscale balances received from the vault in *getVirtualParameters* before calling *_calculateCurrentValues*.
- Document that *calculateCurrentValues* expects upscaled balances.

Status: Fixed.

Update from client: Fixed in commit 7a39ce7daa5844535e7b69ec0ceae193a4d67258.

F10: [Low] GyroThreePool.sol allows AssetManagers to be set at construction

Context: GyroThreePool.sol#L58

Description:

GyroThreePool.sol allows AssetManagers to be set, **GyroTwoPool.sol** does not allow this and the documentation does not specify their use. AssetManagers are used in the Balancer Protocol to invest funds from the pool. If they are going to be used, this should be correctly documented.

Recommendation:

- Set AssetsManagers to zero as done by **GyroTwoPool**.
- Document the use cases of these AssetManagers.

Status: Fixed.

Update from client: Fixed in commit fd83c63e3d0871c942de89005d6c62be67b58db3.

F11: [Low] Governance could be lost if wrong address is used in *changeGovernor*

Context: Governable.sol#L22

Description:

If *changeGovernor* is called with a wrong address as an argument, Governance of contracts could be lost.

Recommendation:

Use a two step process for changing the governor. First propose the new governor and after that claim the governance with the proposed address.

Status: Fixed.

Update from client: Fixed in commit adb52bb849d6c3b499706f96efddfad86de5f92a.

F12: [Low] BPT tokens in BalancerExchanger.sol could be taken by anyone

Context: BalancerExchanger.sol#105

Description:

Any BPT token in **BalancerExchanger.sol** can be withdrawn by any user. The *withdraw* function calls *exitPool* using its own balance of BPT tokens. This contract should not hold any BPT tokens, but if it holds any, they can be stolen.

Recommendation:

We recommend a pull pattern to be used as done in the *deposit* function, before calling *exitPool* to transfer BPT tokens from the caller to the contract.

Status: Fixed.

Update from client: Code not used anymore.

F13: [Low] *deregisterPoolId* only deregisters first occurrence of the pool

Context: BalancerPoolRegistry.sol#40

Description:

The *deregisterPoolId* function stops when the first pool with the indicated *poolId* is found. If a pool was registered more than once, only the first occurrence will be deregistered.

Recommendation:

Because of the needs of the registry we would recommend an Enumerable Map like structure be used instead of a *bytes32[]* for storing the *poolIds* for a specific token.

Status: Fixed.

Update from client: Fixed in commit dfc67845c5f5767be00e30faa70095a0789721ae.

F14: [Low] Disparity between values used in order and values transferred to the reserve

Context: Motherboard.sol#L53

Description:

If the argument *assets* used in the *mint* function contains repeated tokens, the *order* created will not reflect this, only the first version of the *asset* will be accounted for. This could cause issues because all of the accounting and computations of the amounts of tokens to mint are based on the *order*, while the tokens to be transferred to the reserves are defined by the argument *assets*.

Recommendation:

Use a different approach where *order* correctly reflects the values in *assets*.

Status: Fixed.

Update from client: Fixed in pull request #74.

F15: [Low] *getReserveState* can revert without emitting message

Context: ReserveManager.sol#43

Description:

If *options.includeIdealWeight* is *true*, but *options.includePrice* is *false*. The value of *vaultInfo.price* will be 0 causing *returnsSum* to be 0. Because *returnsSum* is 0 this line will revert the transaction.

Recommendation:

Require the value of *options.includePrice* to be true if *options.includeIdealWeight* is true. In the same way as *options.includeCurrentWeight*.

Status: Fixed.

Update from client: Fixed in commit 378c4ffa35263b8bcac9f82d5aa095fab7be9308.

F16: [INFO] Duplicated code in NewtonSqrt.sol file

Context: NewtonSqrt.sol

Description:

All the content of this file is duplicated in **GyroPoolMath.sol**.

Recommendation:

Remove **NewSqrt.sol** file.

Status: Fixed.

Update from client: Fixed in commit 6477d7d3310fac1489f46e0923e0f580334bb026.

F17: [INFO] Non uniform use of term names in comments

Context: GyroPoolMath.sol#L42

Description:

The comments defines terms within formulas using the pattern *term_name = description*. These can then be used within the formula in the comment. In the case of *totalBpt* the *term_name* is *bpt* in the definition but *totalBpt* in the formula.

Recommendation:

Use *bpt* in the formula to keep *term_names* and description uniform.

Status: Fixed.

Update from client: Fixed in commit d8fe54d67f4076ab757657ae4bc1f4a8c74c73b9.

F18: [INFO] Division before multiplication

Context: GyroPoolMath.sol#L37, GyroPoolMath.sol#L63

Description:

In both *_calcAllTokensInGivenExactBptOut* and *_calcTokensOutGivenExactBptIn*, divisions are done before multiplications, doing it in reverse order may improve precision. Our understanding is that this is done in this way because it results in $N + 1$ operations being performed, where N is the number of assets. Whereas, if multiplications are done first, the number of operations is $2N$. When this is used in the **GyroTwoPool**, N is 2, so the difference in the number of operations is only 1. In this case, perhaps the gains in precision are worth the extra operation? In any case, it is not a big issue because any rounding done favors the protocol and the losses caused by the loss of precision to the clients may be mitigated by the gas savings.

Status: Fixed.

Update from client: Fixed in commit a976c3d80297c5aefef2ddbc896cca7cc4da5219.

F19: [INFO] Comments are pointing to the wrong section

Context: GyroPoolMath.sol#L268, GyroPoolMath.sol#L289

Description:

Function comments may be pointing to the wrong section.

Recommendation:

Change section pointed to in the comments by *3.1.3*.

Status: Fixed.

Update from client: Fixed in commit 5259965244ae7e8abab00c0e5cbe9188e477c425.

F20: [INFO] Comments are not aligned with the code

Context: GyroCEMMPool.sol#L547, GyroThreePool.sol#473, GyroTwoPool.sol#521

Description:

The comments of *_getDueProtocolFeeAmounts(uint256 previousInvariant, uint256 currentInvariant)* state that this function returns *uint256[]*, but the function instead returns *(uint256,uin256,address,address)*. The comments also mention this function overrides *getDueProcolFeeAmounts* from the parent contract, but this contract does not have a method with the same signature, because the version in the parent contract receives different arguments.

Recommendation:

Update the comment.

Status: Fixed.

Update from client: Fixed in commit c61dcbac9362b8ec877e59ddf2ee02b84a0c33d1.

F21: [INFO] Unused functions in GyroCEEMMath.sol

Context: GyroCEMMMath.sol#L88, GyroCEMMMath.sol#L52, GyroCEMMMath.sol#L100, GyroCEMMMath.sol#L165

Description:

Functions *validateNormed*, *validateParams* and *validateDerivedParams* are internal functions, but are not used in the code. Calls to these functions are commented out in the **GyroCEEMPool.sol** constructor. The Function *mkDerivedParams* is not used either.

Recommendation:

Use these functions to validate the constructor's arguments. If this is not necessary, remove these functions.

Status: Fixed

Update from client: The functions have been removed.

F22: [INFO] Typo in comments

Context: GyroCEEMMath.sol#L399, GyroCEEMMath.sol#L448

Description:

Comments for lines 400 and 449 may have a typo. They say *"need to to /dSq in a way so ..."*. From the code it appears that it should be *"need to 1/dSq in a way so ..."*.

Recommendation:

Check comments and update if necessary.

Status: Fixed.

Update from client: Fixed in commit 09ca68a3b4755c7b6b678b9e370149d6d6c1c4d5.

F23: [INFO] Comments are not aligned with the code

Context: GyroCEMMMath.sol#L632

Description:

Comments for function *calcXpXpDivLambdaLambda* say $x' * x'$ is computed, but the code computes

$$x' * x' lambda * lambda$$

and the name indicates this too.

Recommendation:

Review comments and code and update accordingly if they are not aligned.

Status: Fixed.

Update from client: Fixed in commit 1ff1c83c34ac63bf80445a6d9f4ff18c91f486a5.

F24: [INFO] *getSqrtParameters* can return a tuple of uint256 to save gas

Context: GyroTwoPool.sol#L58, GyroTwoPool.sol#L62, GyroTwoPool.sol#L104, GyroTwoPool.sol#L228

Description:

The *sqrtParameters()* function returns a *uint256[]*, but it always returns two values and these values are always passed independently to every function. The signature of this function could be change to return *(uint256, uint256)* to save some gas.

Status: Fixed.

Update from client: Function has been adjusted, as well as *_sqrtParameters()* and *_getVirtualParameters()*. To minimize the amount of refactoring required, this have been refactored to use fixed-width arrays instead of multiple variables. Fixed in commit f4635e1f4e32e0594d919d7423f810861a719314.

F25: [INFO] Duplicated comment in GyroTwoPool.sol

Context: GyroTwoPool.sol#L319, GyroTwoPool.sol#325

Description:

Comments in *_onJoinPool* are duplicated.

Recommendation:

Remove duplicated comments.

Status: Fixed.

Update from client: Fixed in commit c142290aafec96f046afa6915c1f28b78d9c1d1b.

F26: [INFO] Reduced readability for *_onSwapGivenIn* and *_onSwapGivenOut*

Context: GyroTwoPool.sol#L170, GyroTwoPool.sol#L182

Description:

ExtensibleWeightedPool2Tokens.sol contains private definitions for functions *_onSwapGivenIn* and *_onSwapGivenOut*. Because these functions are not virtual they can not be overriden and the new functions in **GyroTwoPool.sol** add an unused argument to change the signature. This makes the codebase error prone.

Recommendation:

We recommend the functions in **ExtensibleWeightedPool2Tokens.sol** be made virtual and then be overridden in **GyroTwoPool.sol**.

Status: Fixed.

Update from client: Fixed in commit 7e91b62b8ba42d29e08a484f3835ba686bf96996.

F27: [INFO] Unused arguments in GyroTwoPool.sol constructor

Context: ExtensibleWeighted2TokenPool.sol#L139

Description:

The constructor for **GyroTwoPool.sol** receives two arguments, the first one is a struct with multiple parameters for creating the pool. From these parameters one can compute the weights of the tokens. In **GyroTwoPools.sol** tokens always have 0.5 weight. Receiving these values in the constructor may cause the pool to be initialized with different weights. This does not affect the workings of the pool because they do not rely on these values, but it may cause external interactions to work unexpectedly.

Recommendation:

Always set weights to 0.5 in the constructor without relying on the arguments.

Status: Fixed.

Update from client: The *_normalizedWeight0* and *_normalizedWeight1* variables are now essentially constants with value 0.5 (they are immutable and not set in the constructor, thus keeping their default values of 0.5). Fixed in commit 630c9ae55aafb10a623f827f946a3b2d6fc0a66a.

F28: [INFO] Typo in commented equation in GyroTwoMath.sol

Context: GyroTwoMath.sol#L166

Description:

The numerator of the most derived equation in the comments for the *_calcOutGivenIn* function should be *'y* instead of *'z*.

Recommendation:

Change *'z* to *'y*.

Status: Fixed.

Update from client: Fixed in commit 9a14fa0bc8174283616a2e7f07b530c86458841c.

F29: [INFO] Number of operations in *_calculateCubicTerms* can be reduced

Context: GyroThreeMath.sol#L70

Description:

In the function *_calculateCubicTerms*, multiple expressions contain $root3alpha * root3alpha$ as a subexpression. Similarly for $balances[0] * balances[1]$. These common subexpressions could be eliminated, reducing the number of operations and resulting in gas savings.

Recommendation:

Change the comments to be updated with the code.

Status: Acknowledged

Update from client: The order of operations is chosen intentionally this way to minimize the amplification of rounding errors. This is relevant for the mb value, for instance. Assume for ease of writing that $balances = [x, y, z]$ and write short $\alpha' := root3Alpha$. Then the variable mb is computed essentially as $((x + y + z) * \alpha') * \alpha'$. The inner multiplication implies an error of $1e-18$, which is then multiplied by α', so that the total error is $1e-18*\alpha'+1e-18 \leq 2e - 18$. In contrast, if we were to pre-compute $\alpha' * \alpha'$ with an error of $1e - 18$, and we would then compute $(x + y + z) * (\alpha' * \alpha')$, leading to an error of $(x + y + z) * 1e - 18 + 1e - 18$, which can potentially be significant. Experimentally, such an error would indeed have a noticeable effect on the final result of the invariant calculation. We agree that a common term $balances[0] * balances[1]$ could be isolated, but we decided not to do this small optimization in the interest of readability.

F30: [INFO] *_calcNewtonDelta* has an unused named argument

Context: GyroThreeMath#L160

Description:

_calcNewtonDelta has an argument called *a*, but this argument is not used in the function. If the argument is not used, it should be removed.

Recommendation:

Check if argument is unnecessary, if yes, remove it.

Status: Fixed.

Update from client: This parameter was historical baggage and have been removed from this method and also from *_runNewtonIteration()*. Fixed in commit 39f2fc60b42dbb1471101aed510473034d49784c.

F31: [INFO] *deposit* and *withdraw* do not allow minimum expected amounts to be specified

Context: BalancerExchanger.sol#L66, BalancerExchanger.sol#105

Description:

The *deposit* and *withdraw* functions call *joinPool* and *exitPool* respectively in a Balancer pool. These processes cannot be reverted, users should be able to specify the minimum amounts that they are expecting. Our understanding is that this contract should not be used directly by any users, therefore this is not supported. However, it can still be used by anyone.

Recommendation:

We recommend an argument to specify the minimum amounts be added and, every time these functions are called from other places within the protocol, the value 0 be passed as a minimum amount. This avoids changing the actual behavior of the protocol.

Status: Fixed.

Update from client: Code is not used anymore.

F32: [INFO] Use of *transferFrom* instead of *safeTransferFrom*

Context: BalancerExchanger.sol#71

Description:

In the *deposit* function in **BalancerExchanger.sol**, *transferFrom* is used instead of *safeTransferFrom*. Different checks are performed. The rest of the code uses *safeTransferFrom* from the OpenZeppelin libraries.

Recommendation:

We recommend *safeTransferFrom* be used to maintain the uniformity of the code.

Status: Fixed.

Update from client: Fixed in commit f22d59c1fa361dbbadf331140c9a14cf37891293.

F33: [INFO] Incorrect casing for variable name in *registerPoolId* function

Context: BalancerPoolRegistry.sol#L33

Description:

Variable name *poolIdsforToken* does not follow the variable naming convention with respect to casing.

Recommendation:

Change variable name to *poolIdsForToken*.

Status: Fixed.

Update from client: Fixed in commit 174171ad9549f1808ae1bfd455d8bf3e96732ba0.

F34: [INFO] Unnecessary storage write

Context: BalancerPoolRegistry.sol#L35

Description:

```
poolIdRegistry[underlyingTokenAddress] = poolIdsforToken;
```

This write is unnecessary because *poolIdRegistry* is a storage pointer, it can be removed to save a storage write.

Recommendation:

Remove indicated line.

Status: Fixed.

Update from client: Fixed in commit bbd6201969934abbba04cc00e2fa246bbb84f946.

F35: [INFO] Defined event is never emitted in VaultSafetyMode.sol

Context: VaultSafetyMode.sol#L35

Description:

In the **VaultSafetyMode.sol** contract, the event *MotherboardAddressChanged(address,address)* is defined but never emitted.

Recommendation:

Remove the event definition if is not going to be used.

Status: Fixed.

Update from client: Fixed in commit 3c10173803728751611f6471b95a1950d74e7af1.

F36: [INFO] Motherboard.sol can *mint* tokens directly to *msg.sender* to save gas

Context: Motherboard.sol#L84

Description:

The *mint* function in **Motherboard.sol** mints tokens to the contract itself, after which it sends the tokens to *msg.sender*. Tokens could be minted to *msg.sender* directly to save gas.

Recommendation:

Call *gydToken.mint* with *msg.sender* as receiver.

Status: Fixed.

Update from client: Fixed in commit 81478a32fd0143db2ac0baa854a43ae6b3f1e3af.

F37: [INFO] *setInitialPrice* can be called multiple times for the same vault

Context: VaultRegistry.sol#L73]

Description:

The *setInitialPrice* function in the **VaultRegistry.sol** contract can be called multiple times for the same vault. The name suggest this function should be only called once per vault.

Recommendation:

Check if *initialPrice* for the specified *vault* is 0 before setting the new value.

Status: Fixed.

Update from client: Fixed in commit eae7032318214a836529cf01ed6ec8d0a3bef961.

F38: [INFO] Vault metadata is still available after deregistering the Vault

Context: VaultRegistry.sol#91

Description:

The *deregisterVault* function does not remove the metadata for the specified *vault*. Even after deregistering, the *vault* metadata for this can be fetched through *getVaultMetadata*.

Recommendation:

Remove *vault* metadata in the *deregisterVault* function.

Status: Fixed.

Update from client: Fixed in commit 95799d35a291b5c995c55181c017f217c375f80f.

5 About Nethermind

Founded in 2017 by a small team of world-class technologists, Nethermind builds Ethereum solutions for developers and enterprises. Boosted by a grant from the Ethereum Foundation in August 2018, our team has worked tirelessly to deliver the fastest Ethereum client in the market. Our flagship Ethereum client is all about performance and flexibility. Built on .NET core, a widespread, enterprise-friendly platform, Nethermind makes integration with existing infrastructures simple, without losing sight of stability, reliability, data integrity, and security

Nethermind is made up of several engineering teams across various disciplines, all collaborating to realize the Ethereum roadmap, by conducting research and building high-quality tools. Teams focus on specific areas of the Ethereum problem space. Each consists of specialists and experienced developers working alongside interns, learning the ropes in the Nethermind Internship Program.

Our mission is to gather passionate talent from around the world, and to tackle some of the blockchain's most complex problems. Nethermind provides software solutions and services for developers and enterprises building the Ethereum ecosystem. We offer security reviews to projects built on EVM compatible chains and StarkNet. We have expertise in multiple areas of the Ethereum ecosystem, including protocol design, smart contracts (written in Solidity and Cairo), MEV, etc. We develop some of the most used tools on Starknet and one of the most used Ethereum clients. Learn more about us at `https://nethermind.io`.

Disclaimer

*This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.*