Security Review - Report
NM-0055: Gyroscope CEMM

NETHERMIND

(Aug 22, 2022)

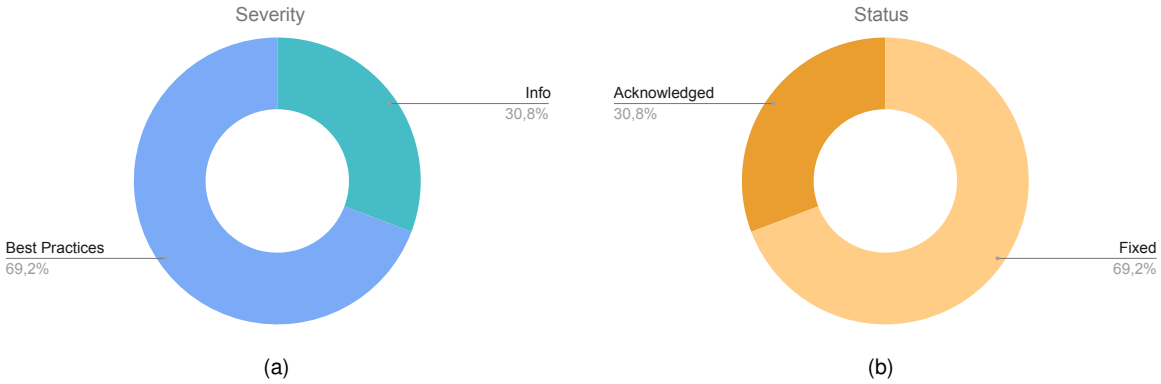
Contents

1 Executive Summary

This document presents the security review performed by Nethermind on the Gyroscope CEMM written in the Solidity Language. Gyroscope proposes an infrastructure for DeFi based on an internal coin, where each unit of this internal coin is backed by 1 USD in collateral. The protocol proposes a novel strategy with multiple lines of defense to maintain price stability.

The primary focus of this audit is on the implementation of the mathematics behind the Constant-Ellipse Market Maker (CEMM) used in the Gyroscope protocol, including an analysis of numerical issues like rounding directions and precision errors. The Gyroscope team provided two detailed papers explaining the mathematical concepts used in the protocol design. The provided papers are titled *The Constant-Ellipse Market Maker (CEMM)* and *CEMM High Precision Calculations*. The Gyroscope team also provided a precision analysis document explaining decisions made by the team to prevent numerical issues.

As an overall assessment, no significant issues were identified. However, we have raised 13 minor issue findings relating to the overall quality and maintainability of the protocol indicated in the figure below. When considering the developer/auditor/user perspective, the quality and volume of inline documentation is sufficient although the code could benefit from improved external technical documentation that specifically focuses on the CEMM component of the protocol.



Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (4), **Best Practices** (9).

Summary of the Audit

Audit Type	Security Review
Initial Report	Jul 18, 2022
Response from Client	Aug 10, 2022
Final Report	Aug 17, 2022
Methods	Manual Review, Automated Analysis
Repository	Gyroscope CEMM
Commit Hash (Initial Audit)	110229131d0caa9ceaaf902ac84ce474147a8a51
Documentation	https://docs.gyro.finance/overview/introduction

2 Contracts

	Contract	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	libraries/GyroPoolMath.sol	192	87	45.3%	32	311
2	libraries/SignedFixedPoint.sol	161	105	65.2%	46	312
3	contracts/cemm/GyroCEMMOracleMath.sol	27	45	166.7%	11	83
4	contracts/cemm/GyroCEMMMath.sol	417	200	48.0%	76	693
5	contracts/cemm/GyroCEMMPoolErrors.sol	15	13	86.7%	5	33
6	contracts/cemm/GyroCEMMPool.sol	345	182	52.8%	86	613
	Total	**1157**	**632**	**54.6%**	**256**	**2045**

3 Summary of Findings

	Finding	Severity	Update
1	Contract exceeding the size limit	Info	Acknowledged
2	Privileged Roles	Info	Fixed
3	Review the rounding strategy for the `GyroPoolMath.sol` constants	Info	Acknowledged
4	Unused state variables	Info	Fixed
5	Application Monitoring Can Be Improved by Emitting More Events	Best Practices	Acknowledged
6	Application using old Solidity version	Best Practices	Fixed
7	Gas optimization opportunities in Functions `mulUpXpToNp(...)` and `mulDownXpToNp(...)`	Best Practices	Fixed
8	Gas optimization opportunities in division functions	Best Practices	Fixed
9	Gas optimization opportunities in functions `divUpMag(...)` and `divUpMagU(...)`	Best Practices	Acknowledged
10	Sanitize input address in `GyroCEMMPool.constructor(...)`	Best Practices	Fixed
11	Unlocked version pragma	Best Practices	Fixed
12	Unnecessary branching in multiplication functions	Best Practices	Fixed
13	Weak validation of token pair in `GyroCEMMPool.onSwap(...)`	Best Practices	Fixed

4 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined two factors: **Likelihood** and **Impact**.

Likelihood is a measure of how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding other factors are also considered. These can include but are not limited to: Motive, opportunity, exploit accessibility, ease of discovery and ease of exploit.

Impact is a measure of the damage that may be caused if the finding were to be exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited: Data/state integrity, loss of availability, financial loss, reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

5 Findings

5.1 General findings

5.1.1 [Info] Privileged Roles

File(s): `contracts/cemm/*`

Description: Smart contracts will often have owner variables to designate the person with special privileges to make modifications to the smart contract. The owner can set fees, incentives, and pause contracts.

Recommendation: This centralization of power needs to be made clear to the users in public-facing documentation.

Status: Acknowledged.

Update from the client: We will ensure to communicate the control aspects of the smart contracts in public-facing documentation.

5.1.2 [Info] Unused state variables

File(s): `contracts/cemm/*`

Description: The following state variables are declared but not used in the CEMM.

```
GyroPoolMath.MIN_NEWTON_STEP_SIZE (libraries/GyroPoolMath.sol)
SignedFixedPoint.MAX_POW_RELATIVE_ERROR (libraries/SignedFixedPoint.sol)
SignedFixedPoint.MIN_POW_BASE_FREE_EXPONENT(libraries/SignedFixedPoint.sol)
```

Recommendation: Ensure that the listed state variables are unused by the protocol outside of the CEMM. If they are unused, then consider removing these state variables.

Status: Fixed.

Update from the client: Fixed in commit 5b702c30fd32e61050d3fb6a290adbaf4a0f0390.

5.1.3 [Best Practices] Application using old Solidity version

File(s): `contracts/cemm/*`

Description: The application is using Solidity `0.7.x`. Since the application makes extensive use of arithmetics, adopting Solidity `0.8.x` appears to be a safer solution as it has built-in overflow and underflow checks. By using version `0.8.x`, there is no need to use the `SafeMath.sol` library. Reference. Moreover, the cost of the `revert` operation is proportional to the string length passed as parameter. Solidity `0.8.x` also introduces the `Error` object, which can be passed as a parameter to the `revert` instruction. Notice that such objects can be defined outside of the contract, allowing the developer to create one single file defining a list of errors for all modules, which can be very convenient. Errors also accept parameters. More information is available here.

Recommendation(s): Consider adopting Solidity `0.8.x`.

Status: Acknowledged.

Update from the client: The underlying Balancer contracts are using 0.7.

5.1.4 [Best Practices] Unlocked version pragma

File(s): `contracts/cemm/*`

Description: The application has pragma `^0.7.0`. Unlocked pragmas could cause the contracts to accidentally be compiled or deployed using an outdated or buggy compiler version.

Recommendation(s): Consider locking pragmas to a specific compiler version.

Status: Fixed.

Update from the client: Fixed solidity version to *0.7.6*. Fixed at commit 6d4fab52364a0a0551d86b3f4c9df928c91c3e22.

5.2 contracts/cemm/GyroCEMMPool.sol

5.2.1 [Info] Contract exceeding the size limit

File(s): contracts/cemm/GyroCEMMPool.sol

Description: The contract GyroCEMMPool.sol is 28,006 bytes, exceeding the EIP-170 limit of 24,577 bytes. There are some actions that can help in reducing the contract size. Whenever we declare a public state variable, the Solidity compiler automatically generates the corresponding getter() function. The contract size can also be optimized by turning modifiers into functions, as shown in the example below:

```
modifier checkStuff() {}
function doSomething() checkStuff {}
```

The function doSomething() can be rewritten without modifiers as:

```
function checkStuff() private {}
function doSomething() { checkStuff(); }
```

Still focusing on optimizing the contract size, we must declare the correct visibility for functions and state variables. Functions or variables that are only called from the outside must be external instead of public. Functions or variables called only from within the contract must be private or internal instead of public. For more info, check this reference.

Recommendation(s): Double-check the code and turn all possible public variables into private variables. Also place the proper visibility of functions in order to optimize the contract size and reduce the deployment cost.

Status: Acknowledged.

Update from the client: We will ensure that the deployed contracts do not exceed the size limit.

5.2.2 [Best Practices] Application Monitoring Can Be Improved by Emitting More Events

File(s): contracts/cemm/GyroCEMMPool.sol

Description: The contract GyroCEMMPool.sol is not emiting events in important state changes. In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transition can be logged, which is beneficial for monitoring the contract, as well as tracking bugs or hacks. Below we present a list of functions that could benefit from event emission to improve protocol monitoring and management.

```
function GyroCEMMPool.constructor(...)
function GyroCEMMPool._onInitializePool(...)
function GyroCEMMPool._onJoinPool(...)
function GyroCEMMPool.onSwap(...)
function GyroCEMMPool._onExitPool(...)
function GyroCEMMPool._updateOracle(...)
```

Recommendation(s): Consider emitting events in the functions listed above.

Status: Fixed.

Update from the client: Events added for each of the suggested functions. Fixed in commit 52b33f04943a374f54a5a63021ec1c6808b4dbc2.

5.2.3 [Best Practices] Sanitize input address in GyroCEMMPool.constructor(...)

File(s): contracts/cemm/GyroCEMMPool.sol

Description: The constructor in GyroCEMMPool is not checking if configAddress is different from address(0x0). Most of the time, this does not represent any risk, since constructors are typically called by initialization scripts. However, validating input parameters is a good programming practice, and can reduce the chances of deploying contracts with wrong input parameters.

Recommendation(s): Consider checking if configAddress is not address(0x0).

Status: Fixed.

Update from the client: Input now checked. Fixed at commit d5a3ee4019891fc03a2dffa862c33befd7392219.

5.2.4 [Best Practices] Weak validation of token pair in `GyroCEMMPool.onSwap(...)`

File(s): `contracts/cemm/GyroCEMMPool.sol`

Description: The function `GyroCEMMPool.onSwap(...)` receives the structure `request` as an input parameter. It assumes that the pair of tokens is properly filled in `SwapRequest.tokenIn` and `SwapRequest.tokenOut`. The value of `tokenInIsToken0` is defined by only comparing if `request.tokenIn==_token0`. Since the pair of tokens is not validated, this can lead to unexpected behavior. A better practice would be to ensure that the pair of tokens really matches the expected one. This check increases the robustness of the function, and prevents human errors. The code is reproduced below.

```
function onSwap(
    SwapRequest memory request,
    uint256 balanceTokenIn,
    uint256 balanceTokenOut
) public virtual override whenNotPaused onlyVault(request.poolId) returns (uint256) {
    bool tokenInIsToken0 = request.tokenIn == _token0;
    ...
}
```

Recommendation(s): Consider validating if the pair of tokens passed as arguments match the expected pair of tokens. Revert otherwise.

Status: Fixed.

Update from the client: Pair validation strengthened. Fixed at commit a5b458d55d909527e262734d816456793504379a.

5.3 `libraries/GyroPoolMath.sol`

5.3.1 [Info] Review the rounding strategy for the `GyroPoolMath.sol` constants

File(s): `libraries/GyroPoolMath.sol`

Description: The constants in `GyroPoolMath.sol` are being truncated, instead of being rounded-up, rounded-down, or rounded towards the closest value. From the provided specification, we could not infer the best behavior for these constants. Below we show the actual definition of constants (according to the code).

```
uint256 private constant SQRT_1E_NEG_1  = 316227766016837933;
uint256 private constant SQRT_1E_NEG_3  = 31622776601683793;
uint256 private constant SQRT_1E_NEG_5  = 3162277660168379;
uint256 private constant SQRT_1E_NEG_7  = 316227766016837;
uint256 private constant SQRT_1E_NEG_9  = 31622776601683;
uint256 private constant SQRT_1E_NEG_11 = 3162277660168;
uint256 private constant SQRT_1E_NEG_13 = 316227766016;
uint256 private constant SQRT_1E_NEG_15 = 31622776601;
uint256 private constant SQRT_1E_NEG_17 = 3162277660;
```

For instance, should the constant `SQRT_1E_NEG_7` be defined as `316227766016837` or rounded-up to `316227766016838`? If we round these constants towards the closest value, some constants have their values modified. The constants with modified values are listed below.

```
uint256 private constant SQRT_1E_NEG_7  = 316227766016838;
uint256 private constant SQRT_1E_NEG_9  = 31622776601684;
uint256 private constant SQRT_1E_NEG_15 = 31622776602;
uint256 private constant SQRT_1E_NEG_17 = 3162277660;
```

Recommendation: This is not an issue, but a design decision that should be made by the Gyro team, according to the mathematical model, and the intended behavior of the application. Our only recommendation is to ensure if these constants are properly defined.

Status: Acknowledged.

Update from the client: The square root calculation is already subject to a small margin of error in the last digit so this error is already contained within that.

5.4 libraries/SignedFixedPoint.sol

5.4.1 [Best Practices] Gas optimization opportunities in Functions mulUpXpToNp(...) and mulDownXpToNp(...)

File(s): libraries/SignedFixedPoint.sol

Description: The functions SignedFixedPoint.mulUpXpToNp(...) and SignedFixedPoint.mulDownXpToNp(...) are computing the variables b1 and b2. Then, those variables are checked if no overflow has occurred. The variable b2 could be calculated after first require(...) statement, and before computing the value of the variable prod2, so if an overflow occurs when computing the value of b1, then the value b2 does not have to be computed. Below we present the code of these functions.

```
function mulUpXpToNp(int256 a, int256 b) internal pure returns (int256) {
    int256 b1 = b / 1e19;
    int256 b2 = b % 1e19;
    int256 prod1 = a * b1;
    if (!(a == 0 || prod1 / a == b1))
        BalancerErrors._require(false, Errors.MUL_OVERFLOW);
    int256 prod2 = a * b2;
    if (!(a == 0 || prod2 / a == b2))
        BalancerErrors._require(false, Errors.MUL_OVERFLOW);
    return prod1 <= 0 && prod2 <= 0 ? (prod1 + prod2 / 1e19) / 1e19 : (prod1 + prod2 / 1e19 - 1) / 1e19 + 1;
}
```

```
function mulDownXpToNp(int256 a, int256 b) internal pure returns (int256) {
    int256 b1 = b / 1e19;
    int256 b2 = b % 1e19;
    int256 prod1 = a * b1;
    if (!(a == 0 || prod1 / a == b1))
        _require(false, Errors.MUL_OVERFLOW);
    int256 prod2 = a * b2;
    if (!(a == 0 || prod2 / a == b2))
        _require(false, Errors.MUL_OVERFLOW);
    return prod1 >= 0 && prod2 >= 0 ? (prod1 + prod2 / 1e19) / 1e19 : (prod1 + prod2 / 1e19 + 1) / 1e19 - 1;
}
```

Recommendations: Consider moving the b2 computation to be performed after the first require(...) statement for gas saving in case an overflow happens when computing the variable b1.

Status: Fixed.

Update from client: Fixed at commit 4d9399c68117991bd6e76059ccdc6fc6f55d293b.

5.4.2 [Best Practices] Gas optimization opportunities in division functions

File(s): libraries/SignedFixedPoint.sol

Description: The division functions SignedFixedPoint.divDownMag(...), SignedFixedPoint.divUpMag(...), SignedFixedPoint.divUpMagU(...), and SignedFixedPoint.divXp(...) are using unnecessary branches. This causes these functions to be more expensive than necessary. For instance, the function SignedFixedPoint.divUpMag(...) has an else branch after the statement if (a == 0), which is not required. Also, the return statement is inside the else branch. The function SignedFixedPoint.divDownMag(...) is represented below.

```
function divDownMag(int256 a, int256 b) internal pure returns (int256) {
    if (b == 0)
        _require(false, Errors.ZERO_DIVISION);

    if (a == 0) {
        return 0;
    } else {
        int256 aInflated = a * ONE;
        if (aInflated / a != ONE)
            _require(false, Errors.DIV_INTERNAL);

        return aInflated / b;
    }
}
```

Recommendation: Consider removing all unnecessary conditional branches.

Status: Fixed.

Update from client: Fixed at commit 4d9399c68117991bd6e76059ccdc6fc6f55d293b.

5.4.3 [Best Practices] Gas optimization opportunities in functions `divUpMag(...)` and `divUpMagU(...)`

File(s): `libraries/SignedFixedPoint.sol`

Description: The functions `SignedFixedPoint.divUpMag(...)` and `SignedFixedPoint.divUpMagU(...)` could be refactored. The line `if (a == 0) return 0` could be moved over the line `if (b < 0)`. That order would save gas if `a==0`, since the function would return and variables wouldn't be reassigned. The functions are reproduced below.

```
function divUpMagU(int256 a, int256 b) internal pure returns (int256) {
        if (b == 0)
            BalancerErrors._require(false, Errors.ZERO_DIVISION);

        if (b < 0) {
            b = -b;
            a = -a;
        }

        if (a == 0) {
            return 0;
        } else {
            if (a > 0) return ((a * ONE - 1) / b) + 1;
            else return ((a * ONE + 1) / b) - 1;
        }
    }
```

```
function divUpMag(int256 a, int256 b) internal pure returns (int256) {
        if (b == 0)
            BalancerErrors._require(false, Errors.ZERO_DIVISION);

        if (b < 0) {
            b = -b;
            a = -a;
        }

        if (a == 0) {
            return 0;
        } else {
            int256 aInflated = a * ONE;
            if (aInflated / a != ONE)
                BalancerErrors._require(false, Errors.DIV_INTERNAL);

            if (aInflated > 0) return ((aInflated - 1) / b) + 1;
            else return ((aInflated + 1) / b) - 1;
        }
    }
```

Recommendation: Consider moving the line `if (a == 0) return 0` over the `if (b < 0 )` to save gas in case of `a==0`.

Status: Fixed.

Update from client: Fixed at commit 4d9399c68117991bd6e76059ccdc6fc6f55d293b.

5.4.4 [Best Practices] Unnecessary branching in multiplication functions

File(s): libraries/SignedFixedPoint.sol

Description: The functions `SignedFixedPoint.mulUpMag(...)` and `SignedFixedPoint.mulUpMagU(...)` are checking if `product` is higher or lower than `0`. The last `else` statement where `product` is equal to `0` is unnecessary. Replacing the last `else` statement with the default `return 0;` statement saves gas (around 44 gas units), since the branch is removed.

```
function mulUpMag(int256 a, int256 b) internal pure returns (int256) {
    int256 product = a * b;
    if (!(a == 0 || product / a == b))
        BalancerErrors._require(false, Errors.MUL_OVERFLOW);

    if (product > 0) return ((product - 1) / ONE) + 1;
    else if (product < 0) return ((product + 1) / ONE) - 1;
    else return 0;
}
```

Recommendation: Review the function and consider removing unnecessary conditional branches.

Status: Fixed.

Update from client: Unnecessary branch removed. Fixed at commit b153816de39cb2cb0a254547be1a757e0bf18c47.

6 Documentation Evaluation

Documenting the code is adding enough information to explain what it does so that it is easy to understand the purpose and the underlying functionality of each file/function/line. Documentation can come not only in the form of a `README.md` but also through inline comments, websites, research papers, videos and external documentation. Besides being a good programming practice, providing proper documentation improves the efficiency of audits. Less time can be spent understanding the protocol and more time can be put towards auditing which improves the efficiency and overall output of the audit.

The documentation for this audit is composed of:

- Gitbook documentation;
- Confidential papers explaining the mathematical model adopted (*The Constant-Ellipse Market Maker (CEMM)* and *CEMM High Precision Calculations*).

In general, the contracts in the scope of this audit have a good amount of inline comments, but the documentation can be improved by adding more technical documents, such as diagrams of interaction, functional/non-functional requirements, and use cases. The `README.md` can present more details about the application. The mathematical model is well presented in the confidential papers provided for the audit. The audited files do not comply with the `NatSpec` standard. The `NatSpec` documentation considers the combination of the tags `@dev`, `@param`, `@notice` and `@return`.

Exceptionally, we noticed insufficient comments in the following functions:

```
function GyroCEMMMath.calcOutGivenIn(...)
function GyroCEMMMath.calcInGivenOut(...)
function SignedFixedPoint.divUpmag(...)
function SignedFixedPoint.sub(...)
```

- The functions `GyroCEMMMath.calcOutGivenIn(...)` and `GyroCEMMMath.calcInGivenOut(...)` are core to the application;
- The function `SignedFixedPoint.divUpMag(...)` does not present comments explaining the way in which upward rounding is achieved in the division;
- The function `SignedFixedPoint.sub(...)` has an incorrect inline comment, possibly copied from the function `SignedFixedPoint.add(...)`.

```
- // Fixed Point addition is the same as regular checked addition
+ // Fixed Point subtraction is the same as regular checked subtraction
```

We also noticed two spelling errors in the file `cemm/GyroCEMMMath.sol`.

```
-     *  Enforces anti-overflow limits on balances and the computed invaraint in the process. */
+     *  Enforces anti-overflow limits on balances and the computed invariant in the process. */
```

```
-     *  See calculation in Section 2.1.2. Calculation is ordered here for precision, but erorr in r is magnified by
 ↪  lambda
+     *  See calculation in Section 2.1.2. Calculation is ordered here for precision, but error in r is magnified by
 ↪  lambda
```

We recommend the following improvements be made to the documentation of this project:

- Adopt the `NatSpec` standard to describe each function (what it does, inputs, outputs);
- Improve the architecture description of the system;
- Create user-friendly documentation describing the logic behind Constant-Ellipse Market Makers;
- Improve the inline comments for functions `GyroCEMMMath.calcInGivenOut(...)` and `calcOutGivenIn(...)`.

7 Test Suite Evaluation

The tests are executed using the command `'brownie test'`. Below we present the tests output, where we can notice the recurrent error message `'fixture QueryProcessor not found'`. In summary 163 tests passed, 39 were skipped, there is 1 warning, and 12 errors in `4,480 seconds`.

```
brownie test
Brownie v1.19.0 - Python development framework for Ethereum

============================================================ test session starts
 ↪  ============================================================
platform linux -- Python 3.9.13, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /home/NM-0055-GYROSCOPE/gyroscope-audit-vaults-main
plugins: eth-brownie-1.19.0, asyncio-0.18.3, xdist-1.34.0, web3-5.29.1, hypothesis-6.27.3, forked-1.4.0,
 ↪  typeguard-2.13.3
asyncio: mode=legacy
collected 214 items

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie
 ↪  --allowUnlimitedContractSize'...

tests/test_decimal_behavior.py .... [  1%]
tests/test_signed_fixed_point.py ....   [  3%]
tests/cemm/test_asymmetric_stretch.py ....  [  5%]
tests/cemm/test_asymmetric_stretch_math.py .sss.....ssss   [ 11%]
tests/cemm/test_cemm_math_implementations_match.py s.sssssss.ss....ss [ 20%]
tests/cemm/test_cemm_oracle_math.py ...   [ 21%]
tests/cemm/test_cemm_pool.py ..EEEEE  [ 24%]
tests/cemm/test_cemm_prec_impl.py .........................[ 37%]
tests/cemm/test_cemm_properties.py ....s  [ 40%]
tests/cemm/test_constant_circle.py ....s  [ 42%]
tests/cemm/test_constant_circle_math.py .sss...ssssss [ 48%]
tests/cemm/test_python_decimals.py ssss   [ 50%]
tests/cemm/test_symmetric_stretch.py .... [ 52%]
tests/cemm/test_symmetric_stretch_math.py .sss.......ss   [ 58%]
tests/cpmmv2/test_gyro_two_math.py .....  [ 60%]
tests/cpmmv2/test_gyro_two_oracle_math.py .   [ 61%]
tests/cpmmv2/test_gyro_two_pool.py ..EEEEEEE  [ 65%]
tests/cpmmv2/test_math_implementations_match.py ......... [ 69%]
tests/cpmmv2/test_two_pool_properties.py .............[ 75%]
tests/cpmmv3/test_gyro_three_math_sensechecks.py .... [ 77%]
tests/cpmmv3/test_gyro_three_pool.py .........[ 81%]
tests/cpmmv3/test_math_implementations_match.py ......[ 84%]
tests/cpmmv3/test_python_calculateInvariant_match.py ...  [ 85%]
tests/cpmmv3/test_three_pool_properties.py ...... [ 88%]
tests/libraries/test_gyro_pool_math.py ...... [ 91%]
tests/libraries/test_signed_fixed_point.py ...... [ 94%]
tests/libraries/test_signed_fixed_point_rounding.py ..... [ 96%]
tests/libraries/test_signed_fixed_point_unchecked.py .......  [100%]

ERROR at setup of test_pool_reg
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cemm/test_cemm_pool.py, line 37: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_on_initialize
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cemm/test_cemm_pool.py, line 83: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found
```

```
/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_on_join
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cemm/test_cemm_pool.py, line 110: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_on_exit
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cemm/test_cemm_pool.py, line 183: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_swap
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cemm/test_cemm_pool.py, line 261: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_reg
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cpmmv2/test_gyro_two_pool.py, line 26: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_factory
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cpmmv2/test_gyro_two_pool.py, line 39: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_constructor
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cpmmv2/test_gyro_two_pool.py, line 49: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_on_initialize
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cpmmv2/test_gyro_two_pool.py, line 87: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_pool_on_join
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cpmmv2/test_gyro_two_pool.py, line 114: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_exit_pool
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cpmmv2/test_gyro_two_pool.py, line 179: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found

/home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py:85
ERROR at setup of test_swap
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/cpmmv2/test_gyro_two_pool.py, line 245: source code not
 ↪  available
file /home/NM-0051-GYROSCOPE/gyroscope-audit-vaults-main/tests/conftest.py, line 85: source code not available
E   fixture 'QueryProcessor' not found
```

```
warnings summary
../../../venv-3.9/lib/python3.9/site-packages/pytest_asyncio/plugin.py:191
  /home/venv-3.9/lib/python3.9/site-packages/pytest_asyncio/plugin.py:191: DeprecationWarning: The 'asyncio_mode'
  ↪  default value will change to 'strict' in future, please explicitly use 'asyncio_mode=strict' or
  ↪  'asyncio_mode=auto' in pytest configuration file.
config.issue_config_time_warning(LEGACY_MODE, stacklevel=2)

-- Docs: https://docs.pytest.org/en/stable/warnings.html

=========================================================
short test summary info
=========================================================
ERROR tests/cemm/test_cemm_pool.py::test_pool_reg
ERROR tests/cemm/test_cemm_pool.py::test_pool_on_initialize
ERROR tests/cemm/test_cemm_pool.py::test_pool_on_join
ERROR tests/cemm/test_cemm_pool.py::test_pool_on_exit
ERROR tests/cemm/test_cemm_pool.py::test_pool_swap
ERROR tests/cpmmv2/test_gyro_two_pool.py::test_pool_reg
ERROR tests/cpmmv2/test_gyro_two_pool.py::test_pool_factory
ERROR tests/cpmmv2/test_gyro_two_pool.py::test_pool_constructor
ERROR tests/cpmmv2/test_gyro_two_pool.py::test_pool_on_initialize
ERROR tests/cpmmv2/test_gyro_two_pool.py::test_pool_on_join
ERROR tests/cpmmv2/test_gyro_two_pool.py::test_exit_pool
ERROR tests/cpmmv2/test_gyro_two_pool.py::test_swap
==================================
163 passed, 39 skipped, 1 warning, 12 errors in 4480.66s (1:14:40)
==================================
Terminating local RPC client...
```

8 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as

ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at 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.