Security Review Report
NM-0410 Cooler V2

NETHERMIND
SECURITY

(April 4, 2025)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for Cooler V2 contracts. Cooler V2 is a lending system that allows users to borrow **USDS** against **gOHM** collateral without losing the voting capabilities from **gOHM**.

Cooler V2 main features include: **a) perpetual positions**, **b) single unified position per user**, **c) governance-controlled LTV growth through a drip system**, and **d) delegation of the deposited collateral's voting power**.

The audit comprises 1851 lines of solidity code. **The audit was performed using** (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases. **Along this document, we report** five points of attention, where one is classified as **Low** and four are classified as **Informational**. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.

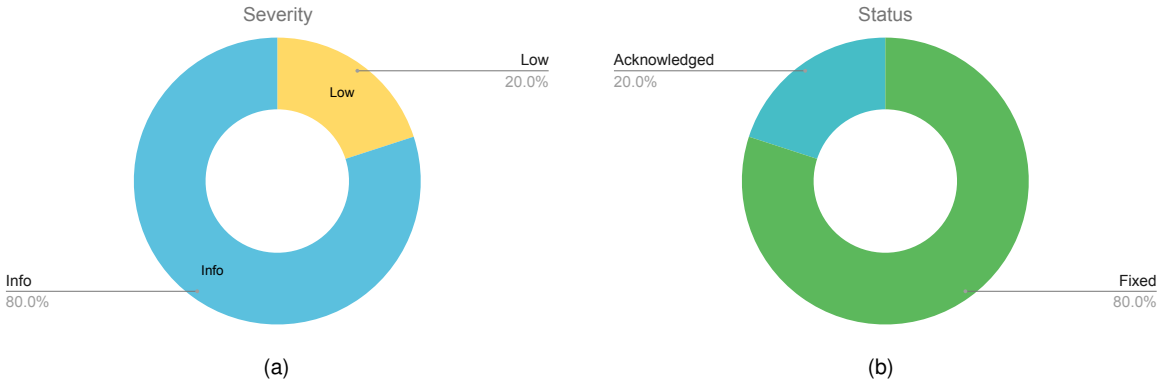


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (4), **Best Practices** (0). **Distribution of status: Fixed** (4), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Final Report	April 4, 2025
Repository	olympus-v3
Commit	ff0ad35b6bba05ddb3ed0efe912bcc6e651d135e
Final Commit	a2ba2de9f17e9901b4510e4f1984c65f114c3a02
Documentation	Docs
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/libraries/CompoundedInterest.sol	17	6	35.3%	4	27
2	src/libraries/SafeCast.sol	16	1	6.2%	3	20
3	src/policies/utils/PolicyEnabler.sol	29	63	217.2%	19	111
4	src/policies/utils/RoleDefinitions.sol	4	4	100.0%	1	9
5	src/policies/utils/PolicyAdmin.sol	25	12	48.0%	7	44
6	src/policies/cooler/MonoCooler.sol	683	235	34.4%	167	1085
7	src/policies/cooler/CoolerTreasuryBorrower.sol	88	32	36.4%	23	143
8	src/policies/cooler/CoolerLtvOracle.sol	159	54	34.0%	40	253
9	src/policies/interfaces/cooler/ICoolerLtvOracle.sol	53	46	86.8%	21	120
10	src/policies/interfaces/cooler/ICoolerTreasuryBorrower.sol	16	25	156.2%	8	49
11	src/policies/interfaces/cooler/IMonoCooler.sol	192	241	125.5%	56	489
12	src/external/cooler/DelegateEscrow.sol	55	25	45.5%	18	98
13	src/external/cooler/DelegateEscrowFactory.sol	49	20	40.8%	14	83
14	src/modules/DLGTE/IDLGTE.v1.sol	64	72	112.5%	23	159
15	src/modules/DLGTE/OlympusGovDelegation.sol	339	86	25.4%	65	490
16	src/modules/DLGTE/DLGTE.v1.sol	62	25	40.3%	17	104
	Total	**1851**	**947**	**51.2%**	**486**	**3284**

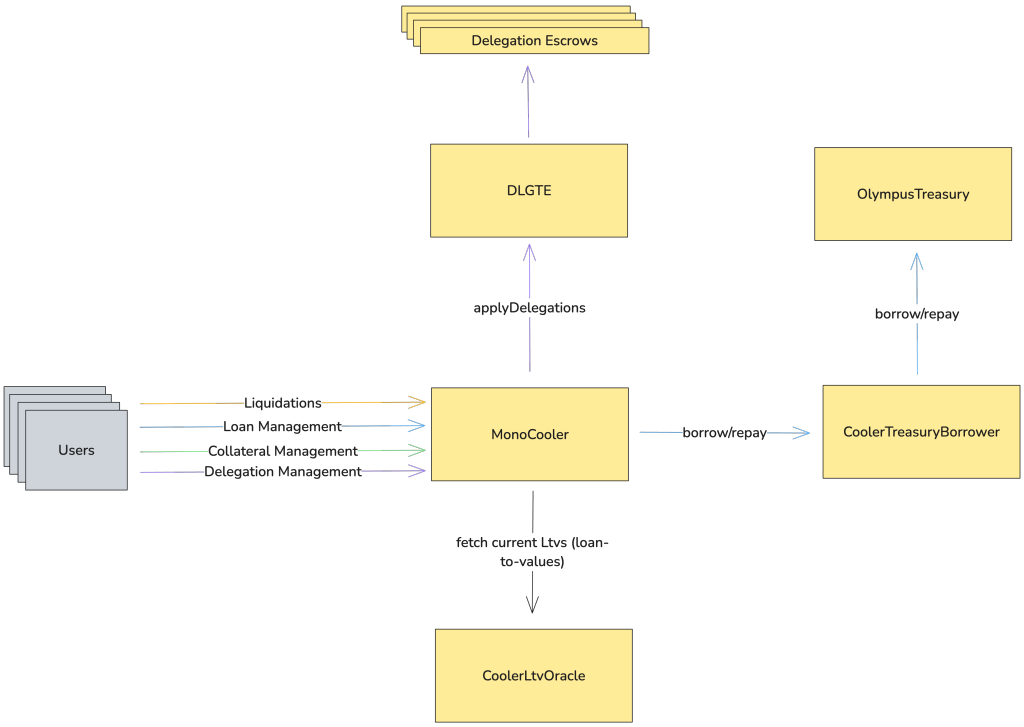
3 Summary of Issues

	Finding	Severity	Update
1	Users may receive slightly more tokens than intended due to rounding in borrow	Low	Fixed
2	Borrowing doesn't always increase debt	Info	Fixed
3	Debt with Treasury is not updated on liquidation	Info	Fixed
4	Liquidation can be delayed by front running call to `applyUnhealthyDelegations(...)`	Info	Fixed
5	Olympus Treasury may lose value due to rounding in `SUSDS.deposit(...)`	Info	Acknowledged

4 System Overview

The Cooler V2 is a lending system that allows users to borrow **USDS** against **gOHM** collateral without losing **gOHM**'s voting power. It allows **gOHM** holders to maximize their earnings on **gOHM** by applying their own strategies. The system is composed by four main components:

- **MonoCooler**
- **CoolerTreasuryBorrower**
- **CoolerLtvOracle**
- **Delegation System**



4.1 MonoCooler

MonoCooler is the main lending contract that manages user positions. This contract handles borrows, repayments, and liquidations. It keeps track of the user's collateral and debt. The main actions users can execute through this contract include:

- **addCollateral(...)**: Users can add **gOHM** as collateral through this function. Users can also change the delegations on their collateral, including the new added value. Any user can add collateral on behalf of another user; however, if delegations are changed, the user must be authorized.
- **withdrawCollateral(...)**: Users can remove their collateral through this function. Only the user or authorized users can withdraw collateral from an account. Changing the delegation of the remaining collateral through this function is also possible. The remaining collateral for an account must be enough to keep their debt at or below the **maxOriginationLtv**. Users can only withdraw undelegated collateral, so they need to adjust delegations in order to withdraw their **gOHM**.
- **borrow(...)**: Users can borrow the debt token through this function. Currently, the debt token is **USDS**. Only the user or authorized users can borrow on behalf of an account. Users can borrow until their debt reaches the current **maxOriginationLtv**.
- **repay(...)**: Users pay debt tokens back to reduce their debt through this function. Any user can do a repayment on behalf of any other user.
- **applyDelegations(...)**: Users can call this function to modify the delegations of their collateral. Only the user or authorized users can modify the delegations.
- **applyUnhealthyDelegations(...)**: This function allows any user to undelegate for other accounts that are in unhealthy positions (can be liquidated). This function exits to execute partial undelegations if an account has delegated to too many addresses.

- **batchLiquidate(...)**: This function is used to liquidate positions that are in an unhealthy state. An account will be unhealthy if its current debt is bigger than its collateral multiplied by the **Liquidation LTV**, account's debt will increase over time due to the borrowing interest applied. When an account is liquidated, its debt is wiped and its collateral is burned. The liquidator will receive **gOHM** as an incentive; the greater the user's debt, the greater the incentive the liquidator will receive. The incentive will be taken from the position's collateral before burning.

It is relevant to note that all the amounts for the debt tokens are specified in **WAD** for this contract. Those amounts are properly scaled through the **CoolerTreasuryBorrower** contract.

4.2 CoolerTreasuryBorrower

CoolerTreasuryBorrower connects **MonoCooler** to the **Treasury**'s **sUSDS** reserves. It handles debt management and conversion between **USDS** and **sUSDS**. The conversion is needed because when **Treasury** holds funds it keeps them in **sUSDS** for yield, but loans are provided in **USDS**.

The contract tracks total debt to **Treasury** and ensures all **USDS** operations are properly converted when interacting with **Treasury**'s **sUSDS** holdings.

The contract has two main functions:

- **borrow(...)**: This function will be called by **MonoCooler** to get funds from the **Treasury**. **MonoCooler** will specify the amount in **WAD**, and this contract will convert into the needed amount and get the funds from the **Treasury**.
- **repay(...)**: The function will be called by **MonoCooler** to repay its debt with the **Treasury** contract. This contract will take its current balance of debt token send the funds back to the **Treasury**, the debt in **Treasury** will be updated accordingly.

4.3 CoolerLtvOracle

CoolerLtvOracle controls how much **USDS** can be borrowed against **gOHM**. Rather than using external market prices, it uses two key thresholds:

- **Origination LTV**: Maximum borrow amount for new loans.
- **Liquidation LTV**: Higher threshold amount that triggers liquidations.

The **Origination LTV** can only increase over time through a linear "drip" system - this means each increase must specify a target value and the time period to reach it. The **Liquidation LTV** is set as a premium above the **Origination LTV**. Reducing the current **Origination LTV** is not possible.

The **Origination LTV** value is expected to be set conservatively, increasing together with the value of **gOHM** but not surpassing it.

4.4 Delegation system

DLGTE manages governance power of **gOHM** while it's being used as collateral in **MonoCooler**. Each account's **gOHM** can be split between multiple delegates through individual escrow contracts, allowing governance participation even while the **gOHM** is locked.

The module tracks balances per policy (like **MonoCooler**) and per user, ensuring one policy can't withdraw **gOHM** deposited by another. Users can delegate to up to ten addresses by default (governance can whitelist a contract to delegate to more than this), with each delegate getting their own escrow contract.

Users cannot interact directly with the **DLGTE** module; they need to do it through the different policies, like the **applyDelegations(...)** function in **MonoCooler**.

5 Risk Rating Methodology

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

Likelihood measures 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 an attacker exploits the finding. 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 to Data/state integrity, loss of availability, financial loss, and 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 Security 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 pass to the client formally;

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.

6 Issues

6.1 [Low] Users may receive slightly more tokens than intended due to rounding in borrow

File(s): src/policies/cooler/CoolerTreasuryBorrower.sol

When a user borrows USDS, the MonoCooler contract invokes the borrow(...) function in CoolerTreasuryBorrower. To facilitate the transfer of the requested USDS amount, the contract first calls SUSDS.previewWithdraw(amountInWad) to calculate the equivalent amount of sUSDS to withdraw from the treasury's reserves. This value is rounded up by previewWithdraw(...) to ensure that at least the requested USDS amount is covered.

After withdrawing the calculated sUSDS amount, the contract calls SUSDS.redeem(...), which transfers the corresponding USDS to the user:

```
function borrow(
    uint256 amountInWad,
    address recipient
) external override onlyEnabled onlyRole(COOLER_ROLE) {
    // ...

    // Since TRSRY holds sUSDS, a conversion must be done before funding.
    // Withdraw that sUSDS amount locally and then redeem to USDS sending to the recipient
    uint256 susdsAmount = SUSDS.previewWithdraw(amountInWad);
    TRSRY.increaseWithdrawApproval(address(this), SUSDS, susdsAmount);
    TRSRY.withdrawReserves(address(this), SUSDS, susdsAmount);
    SUSDS.redeem(susdsAmount, recipient, address(this));
}
```

The issue arises because redeem(...) returns all USDS corresponding to the rounded-up sUSDS amount, potentially resulting in the user receiving slightly more than the intended amount of USDS. This excess is not accounted for in MonoCooler's internal records, introducing minor discrepancies.

While the dust amount is negligible under normal conditions (as the sUSDS price closely tracks USDS at 1:1), the discrepancy could become more significant if the sUSDS price deviates from USDS.

Recommendation(s): Consider using sUSDS.withdraw(...) instead of sUSDS.redeem(...) which will transfer the exact amount of USDS back to the user.

Status: Fixed.

Update from the client: Fixed in commit 66acf6.

6.2 [Info] Borrowing doesn't always increase debt

File(s): src/policies/cooler/MonoCooler.sol

Description: Calculation of debt is always done by rounding up except when borrowed. This results in the scenario where someone can borrow 1 wei without increasing their debt.

```
    function borrow(
        uint128 borrowAmount,
        address onBehalfOf,
        address recipient
    ) external override returns (uint128 amountBorrowed) {
        // ...

        // Sync global debt state when borrowing
        ...

        // @audit debt is calculated by rounding down
        // don't round up the debt when borrowing.
        uint128 currentDebt = _currentAccountDebt(
            _accountDebtCheckpoint,
            aState.interestAccumulatorRay,
            gStateCache.interestAccumulatorRay,
            false
        );

        // ...
    }
```

Recommendation(s): Consider rounding up when calculating current debt in the `borrow(...)` function.

Status: Fixed.

Update from the client: Fixed in commit e54b3f.

6.3 [Info] Debt with Treasury is not updated on liquidation

File(s): src/policies/cooler/CoolerTreasuryBorrower.sol

Description: The `MonoCooler` contract facilitates lending of assets sourced from the Olympus Treasury. The `CoolerTreasuryBorrower` contract interacts with the treasury to withdraw assets during borrowing and return them during repayment. In both the borrow and repay operations, the `CoolerTreasuryBorrower` updates its recorded debt with the treasury accordingly.

However, when a liquidation occurs, the user's debt is cleared within `MonoCooler`, but the corresponding debt recorded in the treasury remains unchanged. This creates an inconsistency between the actual state of loans and the debt values tracked by the treasury.

Such inconsistencies may lead to accounting discrepancies and could affect any processes or policies that rely on the accuracy of debt data maintained by the treasury.

Recommendation(s): Consider implementing a mechanism to update the treasury's debt records when a liquidation occurs.

Status: Fixed.

Update from the client: Fixed in commit a2ba2d.

6.4 [Info] Liquidation can be delayed by front running call to `applyUnhealthyDelegations(...)`

File(s): src/policies/cooler/MonoCooler.sol

Description: To liquidate an unhealthy position, the undelegated assets should be enough to cover the collateral to liquidate. The `batchLiquidate(...)` calls `DLGTE.withdrawUndelegatedGohm(...)` with `autoRescindDelegations` as `true` which will loop through the delegations and undelegate assets if required. Although the default value for the maximum number of delegations that can be done at a time is `10`, admin can allow someone to have more than that. This means that if number of delegations is large enough such that looping through all the delegations and undelegating required assets can't be done in a single transaction, the `applyUnhealthyDelegations(...)` can be used to do this manually in multiple calls before calling `batchLiquidate(...)`. The issue is that if the user front runs the call to `applyUnhealthyDelegations(...)` by changing their delegations then the call to `applyUnhealthyDelegations(...)` would revert. This allows to delay liquidating the position.

Recommendation(s): Consider adding a mechanism to not allow users with unhealthy positions to change their delegations such that to avoid getting liquidated.

Status: Fixed.

Update from the client: Fixed in commit 91d0c6.

6.5 [Info] Olympus Treasury may lose value due to rounding in `SUSDS.deposit(...)`

File(s): src/policies/cooler/CoolerTreasuryBorrower.sol

Description: When a user repays their debt, the `CoolerTreasuryBorrower` contract deposits the repaid `USDS` into the `SUSDS` vault and sends the resulting `SUSDS` tokens to the treasury.

```
function repay() external override onlyEnabled onlyRole(COOLER_ROLE) {
    uint256 debtTokenAmount = _USDS.balanceOf(address(this));
    if (debtTokenAmount == 0) revert ExpectedNonZero();

    // This policy is allowed to overpay TRSRY, in which case it's debt is set to zero
    // and any future repayments are just deposited. There are no 'credits' for overpaying
    uint256 outstandingDebt = TRSRY.reserveDebt(_USDS, address(this));
    uint256 delta;
    if (outstandingDebt > debtTokenAmount) {
        unchecked {
            delta = outstandingDebt - debtTokenAmount;
        }
    }
    TRSRY.setDebt({
        debtor_: address(this),
        token_: _USDS,
        amount_: delta
    });

    _USDS.safeApprove(address(SUSDS), debtTokenAmount);
    // @audit - This function will round down the amount of `SUSDS` to deposit
    SUSDS.deposit(debtTokenAmount, address(TRSRY));
}

```

The `SUSDS.deposit(...)` function calculates the number of shares based on the deposited amount but rounds down when issuing shares. This can result in a small loss of value for the receiver (the treasury) when the deposited amount is not perfectly divisible by the share price.

A malicious user could exploit this behavior by borrowing from `MonoCooler` and repaying their debt in multiple small transactions, each causing the treasury to lose up to `1 wei` worth of `SUSDS` per repayment due to rounding. Over a large number of repayments, this could accumulate to a measurable value loss for the treasury.

However, the attacker does not directly benefit from this behavior. Borrowers must be `gOHM` holders, and diminishing the treasury's assets could negatively affect the value of `gOHM`, harming the attacker's own position. Additionally, the treasury's yield-generating strategies are likely to offset such minor losses.

Consider implementing monitoring or rate-limiting mechanisms on MonoCooler repayments to detect or mitigate repeated small repayments that could exploit rounding behavior.

Recommendation(s): Consider implementing monitoring mechanisms on `MonoCooler` repayments to detect repeated small repayments that could exploit rounding behavior.

Status: Acknowledged.

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about the Cooler V2 documentation

Extensive and thorough documentation was provided for the mechanisms of **Cooler V2**. In addition to documentation, the code was properly commented and explained. The Cooler V2 team was available to explain any doubt or question raised by the Nethermind Security team.

8 Complementary Checks

8.1 Compilation Output

```
pnpm run build

> forge-template@1.0.0 build .../olympus-v3
> chmod +x shell/*.sh && ./shell/full_install.sh

*** Installing dependencies using pnpm
Lockfile is up to date, resolution step is skipped
Packages: +152
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

                  Update available! 10.5.2 → 10.7.1.
      Changelog: https://github.com/pnpm/pnpm/releases/tag/v10.7.1
               Run "corepack use pnpm@10.7.1" to update.


Progress: resolved 152, reused 152, downloaded 0, added 152, done

devDependencies:
+ markdownlint-cli 0.42.0
+ prettier 3.3.3
+ prettier-plugin-sh 0.14.0
+ prettier-plugin-solidity 1.1.3
+ prettier-plugin-sort-json 4.0.0
+ solhint 3.4.1
+ solidity-code-metrics 0.0.25

Done in 891ms using pnpm v10.5.2
*** Setting up submodules
*** Running forge install
Updating dependencies in .../olympus-v3/lib
*** Restoring submodule commits
HEAD is now at 5950723 npm package 1.0.0
HEAD is now at 9310e87 Merge pull request #41 from mzhu25/master
HEAD is now at b93cf4b chore: bump to v1.9.5 (#642)
HEAD is now at 49c0e437 4.8.0
HEAD is now at a4954e5 Update README.md
HEAD is now at fadb2e2  Optimize `SignedWadMath` edge case check (#381)
HEAD is now at 864b357 Merge pull request #5 from OlympusDAO/feat/incorporate-audit-fixes
*** Running forge soldeer update
 Running soldeer update

Successfully downloaded surl~1.0.0 the dependency via git
Dependency surl-1.0.0 downloaded!
Writing surl~1.0.0 to the lock file.
Added a new dependency to remappings @surl-1.0.0
*** Running forge build
[] Compiling...
[] Compiling 34 files with Solc 0.8.15
[] Solc 0.8.15 finished in 30.19s
Compiler run successful with warnings:
Warning (3628): This contract has a payable fallback function, but no receive ether function. Consider adding a receive
 ↪  ether function.
 --> src/external/governance/GovernorBravoDelegator.sol:7:1:
  |
7 | contract GovernorBravoDelegator is GovernorBravoDelegatorStorage, IGovernorBravoEventsAndErrors {
  | ^ (Relevant source part starts here and spans across multiple lines).
Note: The payable fallback function is defined here.
  --> src/external/governance/GovernorBravoDelegator.sol:83:5:
   |
83 |     fallback() external payable {
   |     ^ (Relevant source part starts here and spans across multiple lines).
```

```
Warning (3628): This contract has a payable fallback function, but no receive ether function. Consider adding a receive
↪  ether function.
 --> src/external/governance/Timelock.sol:8:1:

  |
8 | contract Timelock is ITimelock {
  | ^ (Relevant source part starts here and spans across multiple lines).
Note: The payable fallback function is defined here.
  --> src/external/governance/Timelock.sol:89:5:
   |
89 |     fallback() external payable {}
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning (2018): Function state mutability can be restricted to pure
  --> src/proposals/OIP_170.sol:18:5:
   |
18 |     function id() public view override returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/proposals/OIP_170.sol:108:5:
    |
108 |     function _validate(Addresses addresses, address) internal override {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/proposals/OIP_XXX.sol:137:5:
    |
137 |     function _validate(Addresses addresses, address) internal override {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/test/modules/RANGE.t.sol:474:5:
    |
474 |     function testCorrectness_viewRange() public {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/test/modules/RANGE.t.sol:502:5:
    |
502 |     function testCorrectness_viewCapacity() public {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/test/modules/RANGE.t.sol:511:5:
    |
511 |     function testCorrectness_viewActive() public {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/test/modules/RANGE.t.sol:520:5:
    |
520 |     function testCorrectness_viewPrice() public {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/test/modules/RANGE.t.sol:531:5:
    |
531 |     function testCorrectness_viewSpread() public {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/test/modules/RANGE.t.sol:542:5:
    |
542 |     function testCorrectness_viewMarket() public {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
   --> src/test/modules/RANGE.t.sol:551:5:
    |
551 |     function testCorrectness_viewLastActive() public {
    |     ^ (Relevant source part starts here and spans across multiple lines).
```

```
Warning (2018): Function state mutability can be restricted to pure
  --> src/test/proposals/ProposalTest.sol:73:5:
   |
73 |     function testProposal_simulate() public {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/proposals/OIP_170.t.sol:74:5:
   |
74 |     function testProposal_simulate() public {
   |     ^ (Relevant source part starts here and spans across multiple lines).

```

8.2 Tests Output

```
forge test --match-path "**/src/test/policies/cooler/**"
[] Compiling...
No files changed, compilation skipped

Ran 6 tests for src/test/policies/cooler/CoolerLtvOracle.t.sol:CoolerLtvOracleTestAccess
[PASS] test_access_setLiquidationLtvPremiumBps() (gas: 17001)
[PASS] test_access_setMaxLiquidationLtvPremiumBps() (gas: 16976)
[PASS] test_access_setMaxOriginationLtvDelta() (gas: 17025)
[PASS] test_access_setMaxOriginationLtvRateOfChange() (gas: 17095)
[PASS] test_access_setMinOriginationLtvTargetTimeDelta() (gas: 17478)
[PASS] test_access_setOriginationLtvAt() (gas: 17557)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 161.04ms (17.39ms CPU time)

Ran 6 tests for src/test/policies/cooler/MonoCoolerAccess.t.sol:MonoCoolerAccessTest
[PASS] test_access_setBorrowPaused() (gas: 20591)
[PASS] test_access_setInterestRateWad() (gas: 17512)
[PASS] test_access_setLiquidationsPaused() (gas: 20621)
[PASS] test_access_setLtvOracle() (gas: 17469)
[PASS] test_access_setMaxDelegateAddresses() (gas: 17561)
[PASS] test_access_setTreasuryBorrower() (gas: 19646)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 161.15ms (17.49ms CPU time)

Ran 1 test for src/test/policies/cooler/MonoCoolerChangedDebt.t.sol:MonoCoolerChangeDebtToken6dpTest
[PASS] test_changeDebt_existingPosition() (gas: 1057075)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 161.55ms (17.91ms CPU time)

Ran 10 tests for src/test/policies/cooler/MonoCoolerLiquidations.t.sol:MonoCoolerComputeLiquidityTest
[PASS] test_computeLiquidity_afterRepayAll() (gas: 743579)
[PASS] test_computeLiquidity_noBorrowsNoCollateral() (gas: 41445)
[PASS] test_computeLiquidity_noBorrowsWithCollateral() (gas: 216529)
[PASS] test_computeLiquidity_overLLTV_cappedToCollateral() (gas: 451164)
[PASS] test_computeLiquidity_withBorrowAboveLLTV() (gas: 425354)
[PASS] test_computeLiquidity_withBorrowAtOLTV() (gas: 418417)
[PASS] test_computeLiquidity_withBorrowOverOLTV() (gas: 420911)
[PASS] test_computeLiquidity_withBorrowUnderOLTV() (gas: 413727)
[PASS] test_computeLiquidity_withBorrow_ltvDrip_overLLTV() (gas: 443475)
[PASS] test_computeLiquidity_withBorrow_ltvDrip_underLLTV() (gas: 443223)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 163.40ms (19.79ms CPU time)

Ran 15 tests for src/test/policies/cooler/CoolerLtvOracle.t.sol:CoolerLtvOracleTestAdmin
[PASS] test_configureDependencies_fail() (gas: 26118)
[PASS] test_configureDependencies_success() (gas: 23599)
[PASS] test_construction_failDecimalsCollateral() (gas: 1056916)
[PASS] test_construction_failDecimalsDebt() (gas: 874725)
[PASS] test_construction_failLLTVPremiumTooHigh() (gas: 145298)
[PASS] test_construction_failMaxLLTVPremiumTooHigh() (gas: 145236)
[PASS] test_construction_success() (gas: 41190)
[PASS] test_requestPermissions() (gas: 8815)
[PASS] test_setLiquidationLtvPremiumBps_failDecrease() (gas: 19238)
[PASS] test_setLiquidationLtvPremiumBps_failMax() (gas: 19112)
[PASS] test_setLiquidationLtvPremiumBps_success() (gas: 33361)
[PASS] test_setMaxLiquidationLtvPremiumBps() (gas: 33880)
[PASS] test_setMaxOriginationLtvDelta() (gas: 25127)
[PASS] test_setMaxOriginationLtvRateOfChange() (gas: 25432)
[PASS] test_setMinOriginationLtvTargetTimeDelta() (gas: 22225)
Suite result: ok. 15 passed; 0 failed; 0 skipped; finished in 165.25ms (21.94ms CPU time)

Ran 9 tests for src/test/policies/cooler/CoolerLtvOracle.t.sol:CoolerLtvOracleTestOLTV
[PASS] test_setOriginationLtvAt_breachDeltaUp() (gas: 47298)
[PASS] test_setOriginationLtvAt_breachMaxOltvRateOfChange() (gas: 50936)
[PASS] test_setOriginationLtvAt_breachMinDateDelta() (gas: 70940)
[PASS] test_setOriginationLtvAt_cannotDecreaseOLTV() (gas: 21852)
[PASS] test_setOriginationLtvAt_flatAtTargetTime() (gas: 46254)
[PASS] test_setOriginationLtvAt_immediate_successUp() (gas: 55655)
[PASS] test_setOriginationLtvAt_increasesAtExpectedRateSecs() (gas: 63552)
[PASS] test_setOriginationLtvAt_increasesAtExpectedRateYear() (gas: 1631957)
[PASS] test_setOriginationLtvAt_keepTheSame() (gas: 39432)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 165.28ms (21.61ms CPU time)
```

```
Ran 18 tests for src/test/policies/cooler/MonoCoolerAdmin.t.sol:MonoCoolerAdminTest
[PASS] test_changingDependencies() (gas: 3327898)
[PASS] test_configureDependencies_failVersions() (gas: 40580)
[PASS] test_configureDependencies_success() (gas: 64254)
[PASS] test_construction_failDecimalsCollateral() (gas: 1064919)
[PASS] test_construction_failLtv() (gas: 177735)
[PASS] test_construction_success() (gas: 129304)
[PASS] test_requestPermissions() (gas: 19959)
[PASS] test_setBorrowPaused() (gas: 34694)
[PASS] test_setInterestRateWad_fail_max() (gas: 51555)
[PASS] test_setInterestRateWad_success() (gas: 66732)
[PASS] test_setLiquidationsPaused() (gas: 34645)
[PASS] test_setLtvOracle_fail_newLLTV_lt_oldLLTV() (gas: 108341)
[PASS] test_setLtvOracle_fail_newOLTV_gt_newLLTV() (gas: 91206)
[PASS] test_setLtvOracle_fail_newOLTV_lt_oldOLTV() (gas: 108395)
[PASS] test_setLtvOracle_success() (gas: 118301)
[PASS] test_setMaxDelegateAddresses() (gas: 564467)
[PASS] test_setTreasuryBorrower_failDecimals() (gas: 1693127)
[PASS] test_setTreasuryBorrower_success() (gas: 1689485)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 167.09ms (23.38ms CPU time)

Ran 4 tests for src/test/policies/cooler/MonoCoolerCollateral.t.sol:MonoCoolerCollateralApplyDelegationsTest
[PASS] test_applyDelegations_fail_notAuthorized() (gas: 196068)
[PASS] test_applyDelegations_fail_zeroDelegate() (gas: 197974)
[PASS] test_applyDelegations_success_onBehalfOf() (gas: 795977)
[PASS] test_applyDelegations_success_self() (gas: 768154)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 167.43ms (23.83ms CPU time)

Ran 1 test for src/test/policies/cooler/MonoCoolerCollateral.t.sol:MonoCoolerCollateralViewTest
[PASS] test_debtDeltaForMaxOriginationLtv() (gas: 542683)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.65ms (538.04µs CPU time)

Ran 20 tests for src/test/policies/cooler/CoolerTreasuryBorrower.t.sol:CoolerTreasuryBorrowerTestBase
[PASS] test_access_borrow() (gas: 17865)
[PASS] test_access_repay() (gas: 17738)
[PASS] test_access_setDebt() (gas: 17693)
[PASS] test_borrow_failZeroAmount() (gas: 16924)
[PASS] test_borrow_failZeroReceiver() (gas: 16928)
[PASS] test_borrow_once() (gas: 192004)
[PASS] test_borrow_twice() (gas: 249315)
[PASS] test_configureDependencies_fail() (gas: 35642)
[PASS] test_configureDependencies_success() (gas: 33683)
[PASS] test_construction_failDecimalsDebt() (gas: 2254462)
[PASS] test_construction_success() (gas: 27933)
[PASS] test_disabled_borrow() (gas: 26725)
[PASS] test_disabled_repay() (gas: 26532)
[PASS] test_disabled_setDebt() (gas: 26566)
[PASS] test_repay_failZeroAmount() (gas: 22165)
[PASS] test_repay_noDebt() (gas: 110265)
[PASS] test_repay_overRepay() (gas: 245379)
[PASS] test_repay_underRepay() (gas: 245347)
[PASS] test_requestPermissions() (gas: 15538)
[PASS] test_setDebt() (gas: 82252)
Suite result: ok. 20 passed; 0 failed; 0 skipped; finished in 2.77ms (1.51ms CPU time)

Ran 14 tests for src/test/policies/cooler/MonoCoolerBorrow.t.sol:MonoCoolerBorrowTest
[PASS] test_borrow_failBadRecipient() (gas: 11433)
[PASS] test_borrow_failPaused() (gas: 28020)
[PASS] test_borrow_failZeroAmount() (gas: 11360)
[PASS] test_borrow_fail_maxBorrow_overOriginationLtv() (gas: 442429)
[PASS] test_borrow_fail_originationLtv() (gas: 229397)
[PASS] test_borrow_notEnoughDebt_single() (gas: 210137)
[PASS] test_borrow_onBehalfOf_fail_noAuthorization() (gas: 14253)
[PASS] test_borrow_onBehalfOf_withAuthorization() (gas: 539547)
[PASS] test_borrow_success_maxBorrow() (gas: 498648)
[PASS] test_borrow_success_maxBorrow_twice() (gas: 406319)
[PASS] test_borrow_success_newBorrow_diffRecipient() (gas: 472518)
[PASS] test_borrow_success_newBorrow_sameRecipient() (gas: 616304)
[PASS] test_borrow_twice_1dayLater() (gas: 660847)
[PASS] test_borrow_twice_immediately() (gas: 655646)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 169.66ms (26.96ms CPU time)
```

```
Ran 1 test for src/test/policies/cooler/MonoCoolerChangedDebt.t.sol:MonoCoolerChangeDebtToken18dpTest
[PASS] test_changeDebt_existingPosition() (gas: 1052971)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.40ms (1.50ms CPU time)

Ran 14 tests for src/test/policies/cooler/MonoCoolerCollateral.t.sol:MonoCoolerWithdrawCollateralTest
[PASS] test_withdrawCollateral_failNoCollateral_noGohm() (gas: 34008)
[PASS] test_withdrawCollateral_failNoCollateral_withGohm() (gas: 208075)
[PASS] test_withdrawCollateral_failNotEnoughCollateral() (gas: 210255)
[PASS] test_withdrawCollateral_failZeroAmount() (gas: 9776)
[PASS] test_withdrawCollateral_failZeroRecipient() (gas: 9883)
[PASS] test_withdrawCollateral_fail_max() (gas: 425042)
[PASS] test_withdrawCollateral_fail_notEnoughUndelgated() (gas: 500275)
[PASS] test_withdrawCollateral_fail_originationLtv() (gas: 461420)
[PASS] test_withdrawCollateral_onBehalfOf_notAuthorized() (gas: 12633)
[PASS] test_withdrawCollateral_onBehalfOf_withAuthorization() (gas: 292807)
[PASS] test_withdrawCollateral_successDifferentRecipient() (gas: 319848)
[PASS] test_withdrawCollateral_successSameRecipient() (gas: 306924)
[PASS] test_withdrawCollateral_success_maxOriginationLtv() (gas: 506314)
[PASS] test_withdrawCollateral_success_withDelegations() (gas: 611631)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 170.71ms (27.98ms CPU time)

Ran 12 tests for src/test/policies/cooler/MonoCoolerLiquidations.t.sol:MonoCoolerApplyUnhealthyDelegations
[PASS] test_applyUnhealthyDelegations_fail_canOnlyUndelegate1() (gas: 705576)
[PASS] test_applyUnhealthyDelegations_fail_canOnlyUndelegate2() (gas: 976648)
[PASS] test_applyUnhealthyDelegations_fail_cannotLiquidate() (gas: 210150)
[PASS] test_applyUnhealthyDelegations_fail_noUndelegations() (gas: 705416)
[PASS] test_applyUnhealthyDelegations_fail_paused() (gas: 27842)
[PASS] test_applyUnhealthyDelegations_fail_tooMuch() (gas: 711479)
[PASS] test_applyUnhealthyDelegations_fromOtherPolicy_alreadyHasEnoughUndelegated() (gas: 6271358)
[PASS] test_applyUnhealthyDelegations_fromOtherPolicy_empty() (gas: 6283948)
[PASS] test_applyUnhealthyDelegations_fromOtherPolicy_lessThanEnough() (gas: 6380484)
[PASS] test_applyUnhealthyDelegations_fromOtherPolicy_undelegateJustEnough() (gas: 6334382)
[PASS] test_applyUnhealthyDelegations_success_oneUndelegation() (gas: 743239)
[PASS] test_applyUnhealthyDelegations_success_twoUndelegations() (gas: 1016182)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 171.56ms (31.78ms CPU time)

Ran 12 tests for src/test/policies/cooler/MonoCoolerRepay.t.sol:MonoCoolerRepayTest
[PASS] test_repay_failBadOnBehalfOf() (gas: 9164)
[PASS] test_repay_failNoApproval() (gas: 411512)
[PASS] test_repay_failNoDebt() (gas: 32949)
[PASS] test_repay_failUnderMinRequired() (gas: 405606)
[PASS] test_repay_failZeroAmount() (gas: 9107)
[PASS] test_repay_success_afterUnhealthy() (gas: 609520)
[PASS] test_repay_success_evenWhenPaused() (gas: 501049)
[PASS] test_repay_success_full() (gas: 547408)
[PASS] test_repay_success_overFull() (gas: 546384)
[PASS] test_repay_success_partialNotEnoughDebt() (gas: 440883)
[PASS] test_repay_success_partialWithDelay() (gas: 552961)
[PASS] test_repay_success_partial_onBehalfOf() (gas: 837977)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 6.14ms (6.37ms CPU time)

Ran 12 tests for src/test/policies/cooler/MonoCoolerCollateral.t.sol:MonoCoolerAddCollateralTest
[PASS] test_addCollateral_complexDelegations() (gas: 900413)
[PASS] test_addCollateral_failTooMuchDelegation() (gas: 214691)
[PASS] test_addCollateral_failZeroAmount() (gas: 9604)
[PASS] test_addCollateral_failZeroOnBehalfOf() (gas: 9623)
[PASS] test_addCollateral_onBehalfOfNoDelegations() (gas: 278305)
[PASS] test_addCollateral_simple() (gas: 264665)
[PASS] test_addCollateral_thenApplyDelegations_onBehalfOf_noAuthorization() (gas: 627447)
[PASS] test_addCollateral_thenApplyDelegations_onBehalfOf_withAuthorization() (gas: 614422)
[PASS] test_addCollateral_thenApplyDelegations_sameUser() (gas: 576300)
[PASS] test_addCollateral_withDelegations_direct() (gas: 573180)
[PASS] test_addCollateral_withDelegations_onBehalfOf_failUnauthorized() (gas: 192920)
[PASS] test_addCollateral_withDelegations_onBehalfOf_withAuthorization() (gas: 647092)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 9.85ms (9.39ms CPU time)

Ran 12 tests for src/test/policies/cooler/MonoCoolerLiquidations.t.sol:MonoCoolerLiquidationsTest
[PASS] test_batchLiquidate_cappedIncentive() (gas: 473616)
[PASS] test_batchLiquidate_emptyDelegationRequests() (gas: 776144)
[PASS] test_batchLiquidate_fail_paused() (gas: 28148)
[PASS] test_batchLiquidate_noAccounts() (gas: 452016)
[PASS] test_batchLiquidate_noOhmToBurn() (gas: 476511)
```

```
[PASS] test_batchLiquidate_oneAccount_canLiquidateAboveMax() (gas: 580108)
[PASS] test_batchLiquidate_oneAccount_noLiquidate() (gas: 454764)
[PASS] test_batchLiquidate_oneAccount_noLiquidateAtMax() (gas: 441522)
[PASS] test_batchLiquidate_twoAccounts_bothLiquidate() (gas: 858576)
[PASS] test_batchLiquidate_twoAccounts_oneLiquidate() (gas: 848302)
[PASS] test_batchLiquidate_twoAccounts_withUndelegations() (gas: 1459533)
[PASS] test_batchLiquidate_twoCoolers_withUndelegations() (gas: 6305852)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 8.70ms (9.46ms CPU time)

Ran 3 tests for src/test/policies/cooler/MonoCoolerAuthorization.t.sol:MonoCoolerAuthorization
[PASS] test_isSenderAuthorized_self() (gas: 12605)
[PASS] test_setAuthorizationWithSig() (gas: 143481)
[PASS] test_setAuthorization_beforeAtAfterDeadline() (gas: 45708)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 202.14ms (62.20ms CPU time)

Ran 18 test suites in 489.39ms (2.06s CPU time): 170 tests passed, 0 failed, 0 skipped (170 total tests)
```

8.3 Automated Tools

8.3.1 AuditAgent

All the relevant issues raised by the AuditAgent have been incorporated into this report. The AuditAgent is an AI-powered smart contract auditing tool that analyses code, detects vulnerabilities, and provides actionable fixes. It accelerates the security analysis process, complementing human expertise with advanced AI models to deliver efficient and comprehensive smart contract audits. Available at https://app.auditagent.nethermind.io.

9 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.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

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.