Security Review Report
NM-0469-VANA-VEVANA-TOKENS

NETHERMIND SECURITY

(March 28, 2025)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for the smart contracts of Vana to support VeVana tokens. In prior audit engagements, the functionality of the code had been reviewed, and this audit is only focused on integration of VeVana ERC20 tokens and their movement in the protocol.

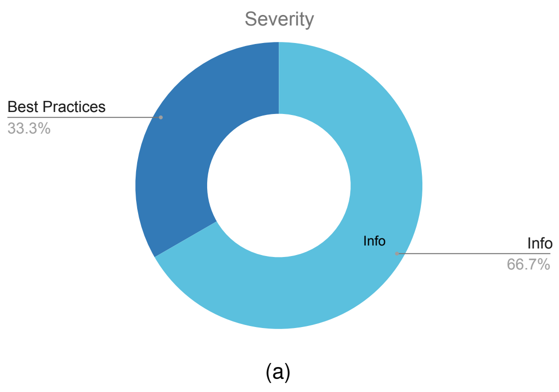
Previously, the economic value of the protocol was only represented in Vana native tokens. With the new changes in the codebase, VeVana ERC20 tokens are integrated into the protocol. This provides the users with the flexibility to use VeVana ERC20 tokens along with Vana native tokens. VeVana token is a wrapper on top of the Vana native token that leverages additional features like voting to the protocol.

VeVana token is implemented as an upgradeable and pausable contract which gives additional control for the protocol owners to add new features to the tokens. On the other hand, it brings the risk of centralization and compromised token owner.

This security review focuses exclusively on the smart contracts listed in Section 2 (*Audited Files*) and scope for interactions with VeVana token contract.

The audited code comprises 1575 lines of code written in the Solidity language, and the audit was performed using (a) manual analysis of the code base and (b) creation of test cases. **Along this document, we report** 3 points of attention, where all three are classified as `Informational` and `Best Practices`. 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 overview of the system. 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.



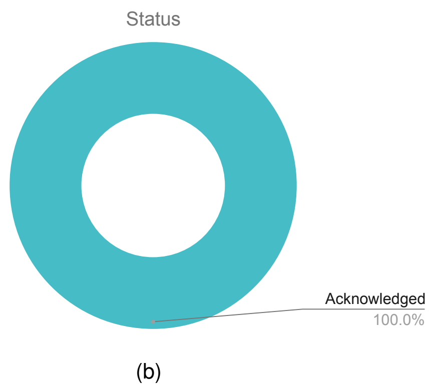


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (2), **Best Practices** (1).
Distribution of status: Fixed (0), **Acknowledged** (3), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Response from Client	Regular responses during audit engagement
Final Report	March 07, 2025
Repository	Vana
Commit (Final)	a9fb959d844b14eb07c4f4566d1e0943caefe1a2
Documentation Assessment	Medium
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	rootEpoch/DLPRootEpochImplementation.sol	265	25	9.4%	67	357
2	root/DLPRootImplementation.sol	455	60	13.2%	124	639
3	root/interfaces/IDLPRoot.sol	82	5	6.1%	10	97
4	rootTreasury/DLPRootTreasuryImplementation.sol	72	4	5.6%	18	94
5	rootTreasury/interfaces/IDLPRootTreasury.sol	14	1	7.1%	2	17
6	rootTreasury/interfaces/DLPRootTreasuryStorageV1.sol	7	6	85.7%	2	15
7	rootCore/DLPRootCoreImplementation.sol	516	34	6.6%	104	654
8	rootCore/interfaces/IDLPRootCore.sol	93	5	5.4%	10	108
9	veVANA/VeVANAProxy.sol	5	1	20.0%	2	8
10	veVANA/VeVANAImplementation.sol	55	6	10.9%	15	76
11	veVANA/interfaces/VeVANAStorageV1.sol	4	6	150.0%	2	12
12	veVANA/interfaces/IVeVANA.sol	7	1	14.3%	4	12
	Total	**1575**	**154**	**9.8%**	**360**	**2089**

3 Summary of Issues

	Finding	Severity	Update
1	Staking on-behalf should source VeVana tokens from the Stake owner's account	Info	Acknowledged
2	migrateStake function will not work as expected	Info	Acknowledged
3	Use Beacon Proxy pattern to keep Rewards and Stakes proxies in Sync	Best Practices	Acknowledged

4 System Overview

VeVana ERC20 tokens are added along with the native Vana tokens for users to perform staking. Users can now stake either native Vana tokens or VeVana ERC20 tokens. Likewise, DLP owners can register DLPs using either of the tokens.

Going forward, the staking rewards will be distributed using VeVana tokens.

VeVana tokens bear voting rights.

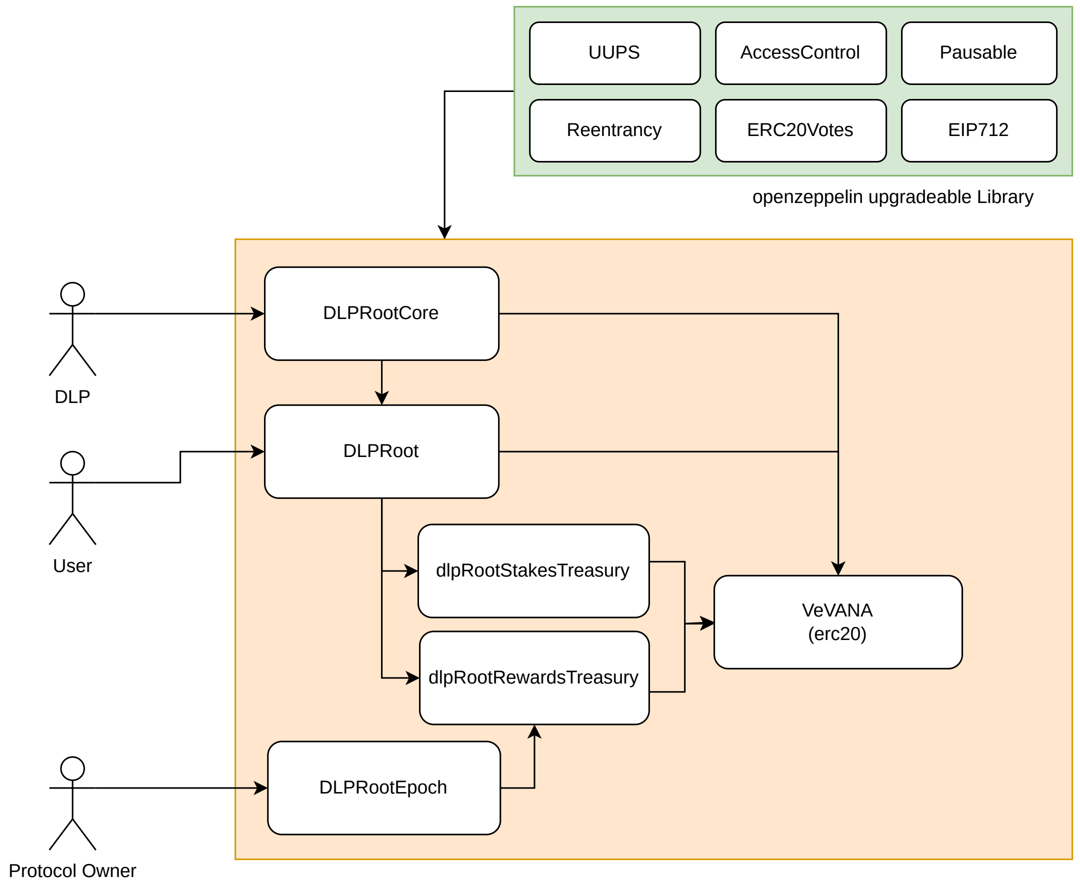


Fig. 2: Vana VeVana interactions

4.1 VeVana Tokens

VeVana is an ERC20 wrapper on top of native Vana tokens. The contract is upgradeable using the UUPS proxy pattern. The token also supports voting functionality and pausing deposit, and withdrawal functions.

4.2 DLP Root

The DLP root contract is extended to support staking functionality using VeVana tokens alongside of Vana native tokens.

4.3 DLP Root Core

DLP root core contract is extended to allow DLP owners to register new DLP using VeVana tokens.

4.4 DLP Epoch

Epoch contract now distributes the rewards in VeVana tokens.

4.5 Root Treasury

The root treasury contract is extended to support integration with VeVana tokens. The contract now allows for deposits in native Vana and VeVana tokens. The contract also withdrawal of value in either native Vana and VeVana tokens.

- Staking Treasury: This treasury instance holds staked funds.
- Rewards Treasury: This treasury instance holds the staking reward funds.

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.

Other factors are also considered when defining the likelihood of a finding. 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.

Other factors are also considered when defining the impact of a finding. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining an issue's likelihood and impact, 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 [Info] Staking on behalf should source VeVana tokens from the Stake owner's account

File(s): DLPRootImplementation.sol

Description: VeVana tokens can be staked by anyone on behalf of another user. These VeVana tokens should be sourced from the stake owner's account and not from the caller of the staking function. In the below two functions, the caller is creating a stake on behalf of stakeOwner. Hence, the stake amount should be sourced from the stakeOwner's account.

```
function createStakeOnBehalf(
    uint256 dlpId,
    address stakeOwner
) external payable override nonReentrant whenNotPaused {
    dlpRootEpoch.createEpochs();

    _createStake(stakeOwner, dlpId, msg.value, block.number, false);//@audit
}

function createStakeOnBehalfWithVeVANA(
    uint256 dlpId,
    address stakeOwner,
    uint256 amount
) external override nonReentrant whenNotPaused {
    dlpRootEpoch.createEpochs();

    _createStake(stakeOwner, dlpId, amount, block.number, false);//@audit
}
```

While stakeOwner was passed as a parameter to _createStake function below, at the time of sourcing the VeVana tokens, the logic is actually sourcing the funds from _msgSender() account instead of the stakeOwner's account.

```
function _createStake(address stakerAddress, uint256 dlpId,
                      uint256 amount, uint256 startBlock,
                      bool isRestake) internal {
    //...
    if (!isRestake) {
        if (msg.value > 0) {
            dlpRootStakesTreasury.depositVana{value: msg.value}();//@audit
        } else {
            IVeVANA veVANA = dlpRootStakesTreasury.veVANA();//@audit
            veVANA.safeTransferFrom(
==>             _msgSender(),
                address(dlpRootStakesTreasury),
                amount
            );
        }
    }
    //...
}
```

As a result, the tokens of the caller will get locked in the stake instead of the stake owner.

Recommendation(s): Take approval from stakeOwner to spend the tokens and pass stakeOwner in safeTransferFrom function. This will also prevent a case where the caller executes a staking on behalf and ends up locking his tokens while the stakeOwner did not deposit the amount into the caller's account.

Status: Acknowledged

Update from the client: The intention of the function is to allow anyone to create a stake in favor of another user using the function caller's tokens. It is like gifting tokens to another user.

6.2 [Info] migrateStake function will not work as expected

File(s): `DLPRootImplementation.sol`

Description: The `migrateStake` function of `DLPRootImplementation` allows the owner to migrate the staked amount from one DLP to another DLP. The process closes the current stake and creates a new stake with the new DLP specified as a parameter.

```
function migrateStake(
    uint256 stakeId,
    uint256 newDlpId,
    uint256 newAmount
) external override nonReentrant whenNotPaused {
    dlpRootEpoch.createEpochs();

    Stake storage stake = _stakes[stakeId];

==> if (newDlpId != stake.dlpId) {
        revert InvalidDlpId();
    }

    if (newAmount > stake.amount) {
        revert InvalidStakeAmount();
    }

    stake.movedAmount = newAmount;
    _closeStake(_msgSender(), stakeId);

    _createStake(_msgSender(), newDlpId, newAmount, stake.startBlock, true);

    emit StakeMigrated(stakeId, stakesCount, newDlpId, newAmount);
}
```

However, the validation of newDlpId expects it to be the same as the stake's dlpId, which defeats the purpose of passing newDlpId as a parameter.

As a result, in the current implementation, the `migrateStake` function will revert if `newDlpId` is not the same as the current stake's `dlpId`.

Recommendation(s): Review the condition for DlpId validation.

Status: Acknowledged

Update from the client: Yes, it is for restaking to another Dlp. However, we block the restaking to a different Dlp currently and only allow restaking to the same Dlp (which is equivalent to partial unstaking).

6.3 [Best Practices] Use Beacon Proxy pattern to keep Rewards and Stakes proxies in Sync

File(s): `DLPRootTreasuryImplementation.sol`

Description: `DLPRootTreasuryImplementation` is an implementation contract that is used by two proxies, one for Rewards and the other for Stakes. In the case of proxy patterns, proxy contracts are the owners of the storage data, and hence each proxy will have their storage.

```
abstract contract DLPRootTreasuryStorageV1 is IDLPRootTreasury {
    IDLPRoot public override dlpRoot;
    IVeVANA public override veVANA;//@audit
}
```

Hence, every time veVana is updated with a new address, it should be updated for both instances of the proxies.

Recommendation(s): Beacon Proxy is a smart contract upgrade pattern where multiple proxies use the same implementation contract, and all the proxies can be upgraded in a single transaction.

Status: Acknowledged

Update from the client: Will consider the recommendation.

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 provide a reference for developers who need to modify or maintain it in the future.

Remarks about Vana documentation

The documentation for the **Vana Smart Contracts** are contained in the README of the project's Github. It is written for developers, presenting each core component of the protocol, including deployment, contracts, functions, and testing. The information is presented in a very concise and technical manner, with well-written explanations and references where necessary. It also features a section explaining development dependencies and instructions on how to build and test the code. Code comments are also of high quality, with explanations for every function describing the purpose, context, and situations where the function will be called. Other than functions, some comments exist in other areas of the code where extra information is necessary, allowing readers to understand the codebase at a faster pace.

8 Test Suite Evaluation

8.1 Compilation Output

```
> npx hardhat compile
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
 ↪  "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for
 ↪  non-open-source code. Please see https://spdx.org for more information.
--> contracts/rootMetrics/interfaces/ERC2771ContextUpgradeableMock.sol


Warning: Source file does not specify required compiler version! Consider adding "pragma solidity ^0.8.24;"
--> contracts/rootMetrics/interfaces/ERC2771ContextUpgradeableMock.sol


Warning: This declaration has the same name as another declaration.
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:331:9:
    |
331 |         address payable foundationWalletAddress
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: The other declaration is here:
  --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:68:5:
   |
68 |     function foundationWalletAddress() external view override returns (address payable) {
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: This declaration shadows an existing declaration.
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:472:14:
    |
472 |         for (uint256 i = 0; i < topDlps.length; ) {
    |              ^^^^^^^^^
Note: The shadowed declaration is here:
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:469:9:
    |
469 |         uint256 i;
    |         ^^^^^^^^^

Warning: This declaration shadows an existing declaration.
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:461:9:
    |
461 |         IDLPRootMetrics.DlpRating[] memory topDlps = topDlpsCustomized(
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:171:5:
    |
171 |     function topDlps(uint256 numberOfDlps) public view override returns (DlpRating[] memory) {
    |     ^ (Relevant source part starts here and spans across multiple lines).
```

```
Warning: This contract has a payable fallback function, but no receive ether function. Consider adding a receive ether
 ↪  function.
  --> contracts/l1Deposit/DepositProxy2.sol:13:1:
   |
13 | contract DepositProxy2 is Proxy {
   | ^ (Relevant source part starts here and spans across multiple lines).
Note: The payable fallback function is defined here.
  --> @openzeppelin/contracts/proxy/Proxy.sol:66:5:
   |
66 |     fallback() external payable virtual {
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Return value of low-level calls not used.
  --> contracts/multisend/MultisendImplementation.sol:57:13:
   |
57 |             recipients[i].call{value: amount}("");
   |             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


Warning: Return value of low-level calls not used.
  --> contracts/multisend/MultisendImplementation.sol:78:13:
   |
78 |             recipients[i].call{value: amounts[i]}("");
   |             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


Warning: Since the VM version paris, "difficulty" was replaced by "prevrandao", which now returns a random number based
 ↪  on the beacon chain.
   --> contracts/multicall3/Multicall3.sol:199:22:
    |
199 |         difficulty = block.difficulty;
    |                      ^^^^^^^^^^^^^^^^


Generating typings for: 128 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 370 typings!
Compiled 126 Solidity files successfully (evm target: paris).
```

8.2 Tests Output

```
> npx hardhat test
DataRegistry
    Setup
       should have correct params after deploy
       should change admin
       Should upgradeTo when owner
       Should upgradeTo when owner and emit event
       Should reject upgradeTo when storage layout is incompatible
       Should reject upgradeTo when non owner
    AddFile
       should addFile
       should addFile multiple times
       should reject addFiles with used fileUrl
       should reject addFile when paused
    Proof
       should addProof, one file, one tee
       should addProof, one file, multiple tee
       should addProof, multiple files, one tee
       should addProof, multiple files, multiple tees
       should reject addProof when paused
    FilePermission
       should addFilePermission, one file, one dlp
       should addFilePermission, one file, multiple dlps #1
       should addFilePermission, one file, multiple dlps #2
       should addFilePermission, multiple files, one dlp
       should addFilePermission, multiple files, multiple dlps
       should reject addFilePermission when non-owner
       should reject addFilePermission when paused
    AddFileWithPermissions
       should addFileWithPermissions, one file, one dlp
       should addFilePermission, one file, multiple dlps #1
       should addFilePermission, one file, multiple dlps #2
       should addFilePermission, multiple files, one dlp
       should addFilePermission, multiple files, multiple dlps
       should reject addFilePermission when non-owner
       should reject addFilePermission when paused

  Deposit
    Setup
       should have correct params after deploy
       should updateMinDepositAmount when owner
       should reject updateMinDepositAmount when non-owner
       should updateMaxDepositAmount when owner
       should reject updateMaxDepositAmount when non-owner
       should updateRestricted when owner
       should reject updateRestricted when non-owner
       Should transferOwnership in 2 steps
       Should reject transferOwnership when non-owner
       Should reject acceptOwnership when non-newOwner
       Should upgradeTo when owner
       Should upgradeTo when owner and emit event
       Should reject upgradeTo when storage layout is incompatible
       Should reject upgradeTo when non owner
    Validators
       should addAllowedValidators when owner
       should reject addAllowedValidators when non-owner
       should addAllowedValidators when owner
       should removeAllowedValidators when owner
       should reject removeAllowedValidators when non-owner
    Deposit
       should deposit when allowed validator #1
       should deposit when allowed validator #2
       should reject deposit when non-allowed validator
       should reject deposit when permission was removed
       should reject deposit when already deposited
       should deposit when restricted = false
```

```
Deposit2
   Setup
      should have correct params after deploy
      should updateMinDepositAmount when owner
      should reject updateMinDepositAmount when non-owner
      should updateMaxDepositAmount when owner
      should reject updateMaxDepositAmount when non-owner
      should updateRestricted when owner
      should reject updateRestricted when non-owner
      Should transferOwnership in 2 steps
      Should reject transferOwnership when non-owner
      Should reject acceptOwnership when non-newOwner
      Should upgradeTo when owner
      Should upgradeTo when owner and emit event
      Should reject upgradeTo when storage layout is incompatible
      Should reject upgradeTo when non owner
   Validators
      should addAllowedValidators when owner
      should reject addAllowedValidators when non-owner
      should addAllowedValidators when owner
      should removeAllowedValidators when owner
      should reject removeAllowedValidators when non-owner
   Deposit
      should deposit when allowed validator #1
      should deposit when allowed validator #2
      should reject deposit when non-allowed validator
      should reject deposit when permission was removed
      should reject deposit when already deposited
      should deposit when restricted = false

 DataLiquidityPool
   Setup
      should have correct params after deploy
      Should pause when owner
      Should reject pause when non-owner
      Should unpause when owner
      Should reject unpause when non-owner
      Should updateFileRewardFactor when owner
      Should reject updateFileRewardFactor when non-owner
      Should updateTeePool when owner
      Should reject updateFileRewardFactor when non-owner
      Should updatePublicKey when owner
      Should reject updatePublicKey when non-owner
      Should updateProofInstruction when owner
      Should reject updateProofInstruction when non-owner
      should change admin
      Should upgradeTo when owner
      Should upgradeTo when owner and emit event
      Should reject upgradeTo when storage layout is incompatible
      Should reject upgradeTo when non owner
   RequestProof
      should requestReward #1
      should requestReward #2
```

```
Multisend
  Setup
     should have correct params after deploy
     Should transferOwnership in 2 steps
     Should reject transferOwnership when non-owner
     Should reject acceptOwnership when non-newOwner
     Should upgradeTo when owner
     Should upgradeTo when owner and emit event
     Should reject upgradeTo when storage layout is incompatible
     Should reject upgradeTo when non owner
  MultisendToken
     should multisendToken to 2 users
     should multisendToken to 500 users
     should reject multisendToken when not enough allowance
     should reject multisendToken when not enough balance
  MultisendVana
     should multisendVana to 2 users
     should multisendVana to 500 users
     should reject multisendVana when invalid amount
  MultisendVanaWithDifferentAmounts
     should multisendVanaWithDifferentAmounts to 2 users, same amount
     should multisendVanaWithDifferentAmounts to 2 users, different amount
     should reject multisendVanaWithDifferentAmounts when not enough funds
     should multisendVanaWithDifferentAmounts to 500 users
     should reject when amounts and recipients arrays have different lengths
  MultisendTokenWithDifferentAmounts
     should multisendTokenWithDifferentAmounts to 2 users with different amounts
     should multisendTokenWithDifferentAmounts to 500 users with random amounts
     should reject when amounts and recipients arrays have different lengths
     should reject when not enough allowance
     should reject when not enough balance

  DLPRoot
  Setup
     should have correct params after deploy
     should pause when maintainer
     should reject pause when non-maintainer
     should unpause when maintainer
     should reject unpause when non-maintainer
     should updateEpochDlpsLimit when maintainer
     should reject updateEpochDlpsLimit when non-maintainer
     should updateEpochSize when maintainer
     should reject updateEpochSize when non-maintainer
     should updateEpochRewardAmount when maintainer
     should reject updateEpochSize when non-maintainer
     should updateMinStakeAmount when maintainer
     should reject updateMinStakeAmount when non-maintainer
     should updateDlpStakersPercentages when maintainer
     should reject updateDlpStakersPercentages when non-maintainer
     should updateMinDlpRegistrationStake when maintainer
     should reject updateMinDlpRegistrationStake when non-maintainer
     should updateDlpEligibilityThresholds when maintainer
     should reject updateDlpEligibilityThresholds when non-maintainer
     should updateStakeWithdrawalDelay when maintainer
     should reject updateStakeWithdrawalDelay when non-maintainer
     should updateTrustedForwarder when maintainer
     should reject updateTrustedForwarder when non-maintainer
     should updateDlpRootMetrics when maintainer
     should reject updateDlpRootMetrics when non-maintainer
     should change admin
     should upgradeTo when owner
     should upgradeTo when owner and emit event
     should reject upgradeTo when storage layout is incompatible
     should reject upgradeTo when non owner
```

```
Dlps - registration
      should registerDlp when stake <  dlpEligibilityThreshold
      should registerDlp with veVANA
      should registerDlp when stake = dlpEligibilityThreshold
      should change eligibility after staking and unstaking
      should registerDlp after epoch1.startBlock
      should registerDlp and add stake
      should registerDlp multiple times
      should reject registerDlp when paused
      should reject registerDlp when stake amount too small
      should reject registerDlp when stakersPercentage too small
      should reject registerDlp when stakersPercentage too big
      should reject registerDlp when already registered
      should reject registerDlp when deregistered
      should reject registerDlp when name already taken
      should reject registerDlp with empty name
      should deregisterDlp when dlp owner
      should reject deregisterDlp when non dlp owner
      should reject deregisterDlp when deregistered
      should updateDlp when dlp owner
      should updateDlp when dlp owner
      should reject updateDlp when not dlp owner
      should reject updateDlp when owner address is zero
      should reject updateDlp when treasury address is zero
      should reject updateDlp when stakers percentage below minimum
      should reject updateDlp when stakers percentage above 100%
      should reject updateDlp when trying to change DLP address
      should updateDlp and update stakersPercentage in next epoch
      should reject updateDlp when paused
      should updateDlp stakerPercentage in the next epoch
      should reject updateDlp when non dlp owner
      should reject updateDlp when invalid stakersPercentage
      should updateDlpVerification when maintainer #true, #registered
      should updateDlpVerification when maintainer #true, #subEligible
      should updateDlpVerification when maintainer #true, #eligible
      should updateDlpVerification when maintainer #false, #registered
      should updateDlpVerification when maintainer #false, #subEligible
      should updateDlpVerification when maintainer #false, #eligible
      should reject updateDlpVerification when non-maintainer
   Epochs
      should createEpochs after the end of the previous one
      should createEpochs after updating rewardAmount
      should createEpochs after updating epochSize
      should createEpochs after long time
      should createEpochsUntilBlockNumber after long time
      should createEpochsUntilBlockNumber with limit
      should createEpochsUntilBlockNumber just until current block number
      should create epochs with no active dlps
      should createEpochs with one registered dlp #1
      should createEpochs after dlpStakersPercentage changes
      should createEpochs with multiple registered dlps #1
      should createEpochs with multiple registered dlps #2
      should createEpochs with multiple registered dlps #3
      should createEpochs with multiple registered dlps #4
      should createEpochs with multiple registered dlps #5
      should createEpochs after staking
      should createEpochs when 100 dlps and 16  epochDlpsLimit
    - should createEpochs when 1000 dlps and 32  epochDlpsLimit
      should overrideEpoch when maintainer
      should revert overrideEpoch when not maintainer
```

```
Staking
       should createStake and emit event
       should createStakeWithVeVANA and emit event
       should create missing epochs when createStake
       should reject createStake when dlp doesn't exist
       should reject createStake when dlp is deregistered
       should reject createStake when stakeAmount < minStakeAmount
       should createStake multiple times, one dlp
       should createStake multiple times, multiple dlps
       should createStake and set lastClaimedIndexEpochId after many epochs
       should createStake when dlp is not verified
       should createStake and not update status if dlp is not verified
       should createStake and update status if dlp is verified
       should createStake and update epochDlpStakeAdjustment after epoch 3
    Close stake
       should closeStake and emit event
       should closeStake multiple stakes in one call
       should closeStake multiple stakes
       should create missing epochs when closeStake
       should reject closeStake when not stake owner
       should reject closeStake when already closed
       should reject closeStake when invalid stake
       should closeStake and update dlp status (eligible -> subEligible)
       should closeStake and update dlp status (eligible -> registered)
       should closeStake and update dlp status (subEligible -> registered)
       should closeStake and keep dlp status (eligible)
       should closeStake and keep dlp status (subEligible)
    Withdraw stake
       should withdrawStake after delay period
       should withdraw multiple stakes in one call
       should create missing epochs when withdrawStake
       should reject withdrawStake when not stake owner
       should reject withdrawStake when already withdrawn
       should reject withdrawStake when not closed
       should reject withdrawStake when withdrawal delay not passed
       should withdraw stake after delay update
    TopDlps
       should set topDlps when creating new epoch (dlpsCount = 0,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 1,  epochDlpsLimit = 1)
       should set topDlps when creating new epoch (dlpsCount = 2,  epochDlpsLimit = 2)
       should set topDlps when creating new epoch (dlpsCount = 3,  epochDlpsLimit = 3)
       should set topDlps when creating new epoch (dlpsCount = 16,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 32,  epochDlpsLimit = 32)
       should set topDlps when creating new epoch (dlpsCount = 2,  epochDlpsLimit = 1)
       should set topDlps when creating new epoch (dlpsCount = 3,  epochDlpsLimit = 1)
       should set topDlps when creating new epoch (dlpsCount = 16,  epochDlpsLimit = 1)
       should set topDlps when creating new epoch (dlpsCount = 32,  epochDlpsLimit = 1)
       should set topDlps when creating new epoch (dlpsCount = 1,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 2,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 3,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 16,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 30,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 40,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 50,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 60,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 100,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch (dlpsCount = 200,  epochDlpsLimit = 16)
       should set topDlps when creating new epoch after dlpOwner staking
       should set topDlps when creating new epoch after user staking
       should set topDlps when creating new epoch after unstaking
       should set topDlps when creating new epoch after unstaking #2
       should set topDlps when creating new epoch after registering new DLPs
       should set topDlps when creating new epoch after a DLP deregisters
       should set topDlps when creating new epoch after updating the maximum number of DLPs #updateEpochDlpsLimit
      should set topDlps when creating new epoch #staking, unstaking, registration, deregistration, updateEpochDlpsLimit
```

```
Save epoch DLPs total stakes score
       should saveEpochDlpsTotalStakesScore and emit event
       should reject saveEpochDlpsTotalStakesScore when non-manager
       should reject saveEpochDlpsTotalStakesScore for unregistered dlpId
       should reject saveEpochDlpsTotalStakesScore for future epochs
       should reject saveEpochDlpsTotalStakesScore when score already exists
       should saveEpochDlpsTotalStakesScore for multiple valid scores
       should reject saveEpochDlpsTotalStakesScore when any score in batch is invalid
       should saveEpochDlpsTotalStakesScore for deregistered DLPs past epochs
       should overrideEpochDlpsTotalStakesScore for new score and emit event
       should reject overrideEpochDlpsTotalStakesScore when called by non-maintainer
       should reject overrideEpochDlpsTotalStakesScore for unregistered DLP ID
       should reject overrideEpochDlpsTotalStakesScore for future epochs
       should overrideEpochDlpsTotalStakesScore for existing score and emit event
       should overrideEpochDlpsTotalStakesScore with same value and emit event
       should overrideEpochDlpsTotalStakesScore for zero value and emit event
       should overrideEpochDlpsTotalStakesScore for deregistered DLPs past epochs
    Calculate stake score
       should calculateStakeScore return correct values for 0-86 days
       should calculateStakeScore same block
       should calculateStakeScore for less than one day
       should calculateStakeScore for exactly one day
       should calculateStakeScore for one week
       should calculateStakeScore for 20 days
       should calculateStakeScore for maximum multiplier
       should calculateStakeScore with fractional days
       should calculateStakeScore with zero stake amount
       should calculateStakeScore with small stake amounts
       should calculateStakeScore with large stake amounts
    Claim stakes reward - rewardClaimDelay = 0
       should claimStakesReward
       should reject claimStakesReward when paused
    DLPRootMetrics
      Setup
         should have correct params after deploy
         should change admin
         should updateDlpRoot when maintainer
         should reject updateDlpRoot when non-maintainer
         Should upgradeTo when owner
         Should upgradeTo when owner and emit event
         Should reject upgradeTo when storage layout is incompatible
         Should reject upgradeTo when non owner
      SaveEpochPerformanceRatings
         should saveEpochPerformanceRatings
         should saveEpochPerformanceRatings after epoch.endBlock
         should saveEpochPerformanceRatings after epoch.endBlock, 100 dlps
       - should saveEpochPerformanceRatings for 500 dlps
         should reject saveEpochPerformanceRatings when non-manager
    DLPRootRewardsTreasury
      Setup
         should have correct params after deploy
         should change admin
         should updateDlpRoot when maintainer
         should reject updateDlpRoot when non-maintainer
         should migrate VANA to veVANA with admin role
         should allow to transfer VANA or veVANA after VANA migration
         should not migrate VANA to veVANA with non-admin role
         Should upgradeTo when owner
         Should upgradeTo when owner and emit event
         Should reject upgradeTo when storage layout is incompatible
         Should reject upgradeTo when non owner
      Receive
         should receive
         should receive veVANA when deposit VANA
      Transfer Vana
         should transferVana when owner
         should not transferVana when non-owner
    DLPRootStakesTreasury
      Setup
         should have correct params after deploy
         should change admin
         should updateDlpRoot when maintainer
         should reject updateDlpRoot when non-maintainer
         Should upgradeTo when owner
         Should upgradeTo when owner and emit event
         Should reject upgradeTo when storage layout is incompatible
         Should reject upgradeTo when non owner
      Receive
         should receive
         should receive veVANA when deposit VANA
      Transfer Vana
         should transferVana when owner
         should not transferVana when non-owner

  TeePool
    Setup
       should have correct params after deploy
       should change admin
       Should updateDataRegistry when owner
       Should reject updateDataRegistry when non-owner
       Should updateTeeFee when owner
       Should reject updateTeeFee when non-owner
       Should updateCancelDelay when owner
       Should reject updateCancelDelay when non-owner
       Should multicall
       Should upgradeTo when owner
       Should upgradeTo when owner and emit event
       Should reject upgradeTo when storage layout is incompatible
       Should reject upgradeTo when non owner
    Tee management
       should addTee when owner
       should addTee #multiple tees
       should reject addTee when already added
       should reject addTee when non-owner
       should removeTee when owner #1
       should removeTee when multiple tees
       should reject removeTee when non-owner
       should reject removeTee when not added
    Job
       should requestContributionProof
       should submitJob
       should requestContributionProof #same user multiple files
       should requestContributionProof for same file #multiple users same file
       should requestContributionProof #multiple users multiple files
       should requestContributionProof without bid when teeFee = 0
       should reject requestContributionProof when insufficient fee
       should cancelJob with bid when teeFee != 0
       should cancelJob when multiple jobs #1
       should cancelJob when multiple jobs #2
       should cancelJob without bid when teeFee = 0
       should reject cancelJob before cancelDelay
       should reject cancelJob when not job owner
    Proof
       should addProof when assigned tee #1
       should addProof when assigned tee #2
       should reject addProof when not tee
       should reject addProof when not active tee
       should reject addProof when proof already submitted
       should addProof for multiple files
      Claim
         should claim
         should reject withdraw when not tee
         should reject withdraw when nothing to claim
         should reject claim when already claimed
         should claim multiple times
    End to End
       example 1
```

```
ERC20
  DLPT - basic
     should have correct params after deploy
     Should transferOwnership in 2 steps
     Should reject transferOwnership when non-owner
     Should changeAdmin when owner
     Should reject changeAdmin when non-owner
     Should blockMint when owner
     Should reject blockMint when non-owner
     Should mint when owner
     Should reject mint when non-owner
     Should reject mint when minting is blocked
     Should blockAddress when admin
     Should reject blockAddress when non-admin
     Should unblockAddress when admin #1
     Should reject unblockAddress when non-admin
     Should unblockAddress when admin #2
     Should transfer
     Should reject transfer when blocked
     Should transfer when unblocked
  DLPT - voting
     should delegate
     should have 0 votes when blocked
     should reject delegate when blocked
     should cancel delegate when blocked

Treasury
  Setup
     should have correct params after deploy
     Should transferOwnership in 2 steps
     Should reject transferOwnership when non-owner
     Should reject acceptOwnership when non-newOwner
     Should upgradeTo when owner
     Should upgradeTo when owner and emit event
     Should reject upgradeTo when storage layout is incompatible
     Should reject upgradeTo when non owner
  Receive
     should receive VANA
  Withdraw
     should withdraw token when owner
     should not withdraw token when non owner
     should withdraw VANA when owner
     should not withdraw VANA when non owner
veVANA
  Deployment
     should have the correct name and symbol
     should have the correct decimals
     should have the correct total supply
     should have the correct owner
  Deposit
     should allow depositing
     should emit a Deposited event on deposit
     should reject if the deposit amount is 0
     should reject if the caller is not the owner
  Withdraw
     should allow withdrawing
     should emit a Withdrawn event on withdraw
     should reject if the withdraw amount is 0
     should reject if the withdrawn amount exceeds the balance
     should reject if the caller is not the owner
  Ownership
     should allow transferring ownership
     should emit an OwnershipTransferred event on ownership transfer
     should reject if the caller is not the owner
     should allow renouncing ownership
     should allow anyone to deposit and withdraw after ownership is renounced
  Governance
     should provide the correct voting power to veVANA holders

458 passing (2m)
2 pending
```

8.2.1 Slither

All the relevant issues raised by Slither have been incorporated into the issues described in this report.

8.2.2 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