Security Review Report
NM-0451 Vana DLPRoot Restructure

(February 20, 2025)

Contents

1 Executive Summary

This document outlines the security review performed by Nethermind Security for the Vana DLPRoot smart contract restructure. In a prior audit engagement the general functionality of the code had been reviewed, but due to bytecode size constraints a single contract has been split into multiple separate contracts. In addition to the restructure of the DLPRoot, the following VRCs (vana-request-for-comments) have been implemented:

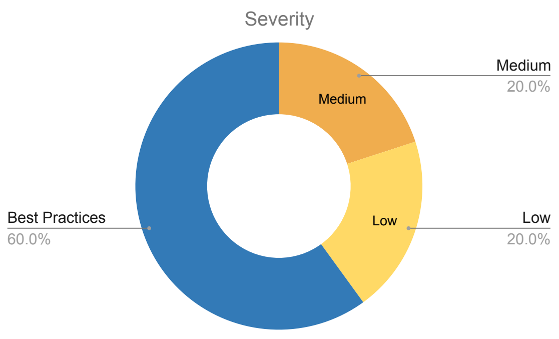
- **VRC-5**: Introducing DLP whitelisting, only allowing certain DLPs to be considered a "top performer" and eligible for rewards.
- **VRC-8**: An adjustment to the staking reward multiplier, to discourage short term staking and prevent dishonest reward claiming.

The existing DLPRoot contract will be upgraded to the new implementation, and storage data from this contract will be taken and transferred to the new contracts. This code review focuses on the newly introduced implementations, but does not include the migration logic itself.

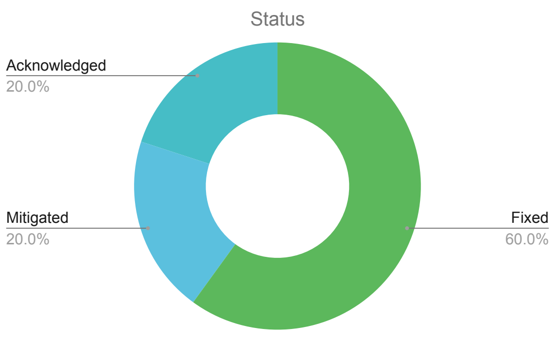
The audited code comprises of 2104 lines of code written in the Solidity language, and the audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract.

Along this document, we report five points of attention, where one is classified as `Medium`, one is classified as `Low`, and three are classified as `Informational` or `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 describes 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 and 9 detail automated tooling used during the audit. Section 10 presents the compilation, tests, and automated tests. Section 11 concludes the document.



(a) distribution of issues according to the severity



(b) distribution of issues according to the status

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

Summary of the Audit

Audit Type	Security Review
Draft Report	February 20, 2025
Final Report	-
Methods	Manual Review, Automated analysis
Repository	`vana-com/vana-smart-contracts`
Initial Commit Hash	`13296575d936d8aa9e987b5d258a8f8764f37ece`
Final Commit Hash	-
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	contracts/rootTreasury/DLPRootTreasuryImplementation.sol	46	2	4.3%	13	61
2	contracts/rootTreasury/DLPRootRewardsTreasuryProxy.sol	5	1	20.0%	2	8
3	contracts/rootTreasury/DLPRootStakesTreasuryProxy.sol	5	1	20.0%	2	8
4	contracts/rootTreasury/interfaces/IDLPRootTreasury.sol	8	1	12.5%	2	11
5	contracts/rootTreasury/interfaces/DLPRootTreasuryStorageV1.sol	5	6	120.0%	2	13
6	contracts/rootEpoch/DLPRootEpochProxy.sol	5	1	20.0%	2	8
7	contracts/rootEpoch/DLPRootEpochImplementation.sol	223	30	13.5%	56	309
8	contracts/rootEpoch/interfaces/IDLPRootEpoch.sol	68	4	5.9%	11	83
9	contracts/rootEpoch/interfaces/DLPRootEpochStorageV1.sol	11	2	18.2%	5	18
10	contracts/rootMetrics/DLPRootMetricsImplementation.sol	502	28	5.6%	101	631
11	contracts/rootMetrics/DLPRootMetricsProxy.sol	5	1	20.0%	2	8
12	contracts/rootMetrics/interfaces/DLPRootMetricsStorageV1.sol	9	6	66.7%	5	20
13	contracts/rootMetrics/interfaces/IDLPRootMetrics.sol	102	1	1.0%	11	114
14	contracts/root/DLPRootImplementation.sol	439	57	13.0%	119	615
15	contracts/root/DLPRootProxy.sol	5	1	20.0%	2	8
16	contracts/root/interfaces/IDLPRoot.sol	80	5	6.2%	10	95
17	contracts/root/interfaces/IDLPRootDeprecated.sol	44	1	2.3%	5	50
18	contracts/root/interfaces/DLPRootStorageV1.sol	36	10	27.8%	11	57
19	contracts/rootCore/DLPRootCoreProxy.sol	5	1	20.0%	2	8
20	contracts/rootCore/DLPRootCoreImplementation.sol	393	43	10.9%	82	518
21	contracts/rootCore/interfaces/DLPRootCoreStorageV1.sol	16	1	6.2%	5	22
22	contracts/rootCore/interfaces/IDLPRootCore.sol	92	5	5.4%	10	107
	Total	**2104**	**208**	**9.9%**	**460**	**2772**

3 Summary of Issues

	Finding	Severity	Update
1	Prevent protocol interactions mid-migration	Low	Unresolved
2	Storage mapping `dlpNameToId` is inconsistently used	Low	Unresolved
3	Unnecessary ERC2771 functions in `DLPRootMetrics`	Info	Unresolved
4	Updating `epochSize` requires an upgrade of `DLPRootMetrics`	Info	Unresolved
5	`createEpochs` can be used instead of `createEpochsWithBlockNumber`	Best Practices	Unresolved

4 Protocol Overview

The restructure of the DLPRoot contract splits it into four separate contracts, each of which is described below:

DLPRoot: This is the original contract handling all aspects of staking, including staking, unstaking, withdrawals, and tracking related data. It also includes features for migrating stakes from a previous implementation and allows claiming rewards.

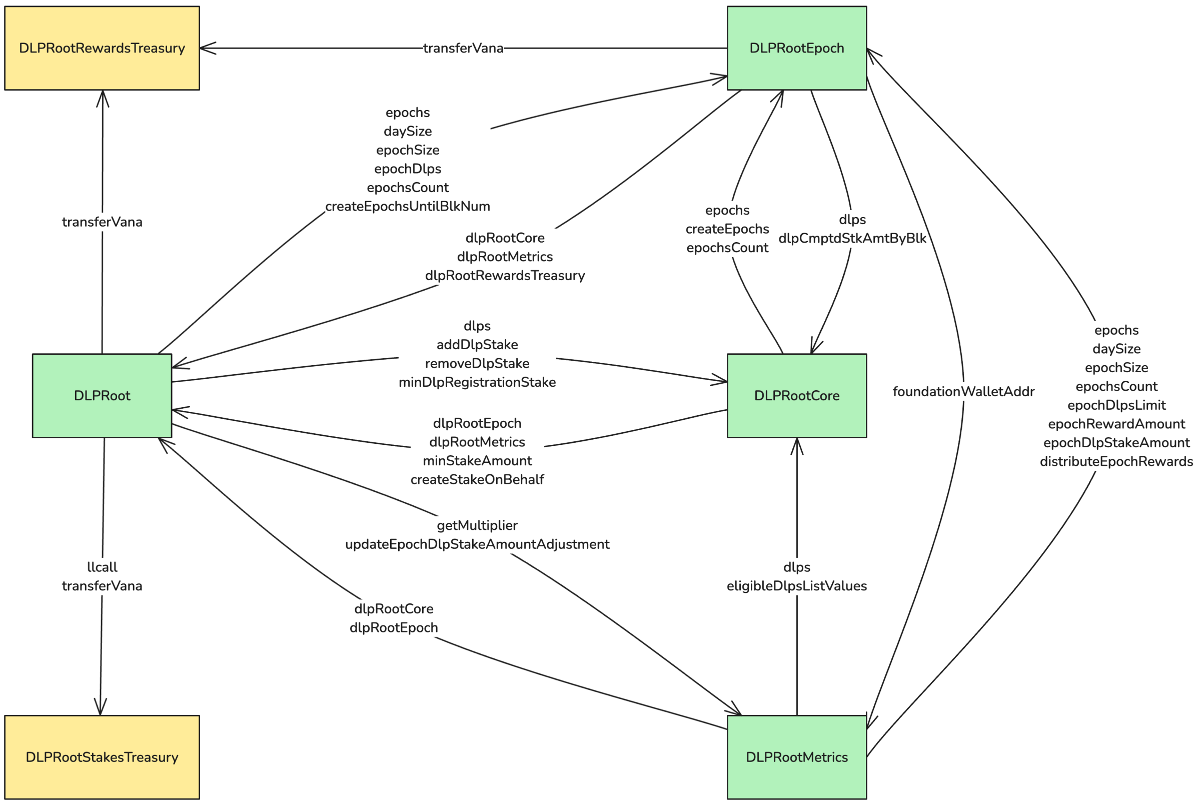
DLPRootEpoch: This contract manages all aspects of epochs, including configuration, updating epoch limits and sizes, and distributing rewards based on total stake scores. It also handles epoch finalization.

DLPRootCore: This contract handles DLP registration, allowing DLP owners to register, deregister, update registration data, and track DLP stakes. It is crucial for tracking top-performing DLPs eligible for special rewards.

DLPRootMetrics: This contract interacts with the DLP Root Epoch and calculates top-performing DLPs, estimates DLP reward percentages, tracks multipliers for stake scores, and allows admins to apply performance ratings for each epoch.

DLPRootRewardsTreasury: This contract stores assets designated for rewards from staking gains. It ensures that reward distributions are handled securely and transparently.

DLPRootStakesTreasury: This contract stores assets that users put up for staking, returning them upon withdrawal. It ensures proper management of staked assets throughout the process.



DLP Root Structure

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] Prevent protocol interactions mid-migration

File(s): `*.cairo`

Description: If the migration process is not atomic (the migration is executed in a single transaction) it will be possible to interact with the protocol while some of the new contracts (`DLPRootCore`, `DLPRootEpoch`) are deployed but not properly initialized with the correct state.

For the `DLPRootCore` it may be possible for users to register certain DLP addresses before the migration, with fields like `name`, `owner` and `treasury` being controlled by the person registering, even if they're not the real owner of the DLP. This would only be possible in the short time window between the `DLPRootCore` contract being deployed, and the storage migration.

For the `DLPRootEpoch` a malicious user could call the `createNewEpochsUntilBlockNumber` function with some block number such that new epochs are created. If this is called before the existing epochs have been migrated, then the contract will consider itself to be on epoch0, with the last epoch block being at zero, since the storage has not yet been initialized. By calling this function after deployment but before the existing epochs have been migrated, a malicious user could increase the `epochsCount` which cannot be reversed. If this were to become "out of sync" with the migrated storage data in the other contracts it may cause abnormal protocol behavior.

Recommendation(s): Consider either using an atomic migration process where all changes are applied in one transaction, or to only allow the state changing public functions to be callable after the migration process is complete.

Status: Unresolved

Update from the client:

6.2 [Low] Storage mapping `dlpNameToId` is inconsistently used

File(s): `DLPRootCoreImplementation.cairo`

Description: The mapping `dlpNameToId` allows a DLP to be found using its name, and also is used for uniqueness upon registration of a new DLP. However, when updating a DLP via `updateDlp` the name can be changed directly without this uniqueness check or an empty string check. This makes it possible for two DLPs to have the same name. Furthermore, when updating the name of a DLP, the `dlpNameToId` mapping is not updated to reflect this change, so the mapping will always point to the registration name rather than the current name.

Recommendation(s): Consider adding the same name validation to `updateDlp` as there is for `createDlp`, and make updates to the `dlpNameToId` mapping during updates.

Status: Unresolved

Update from the client:

6.3 [Info] Unnecessary ERC2771 functions in `DLPRootMetrics`

File(s): `DLPRootMetrics.cairo`

Description: The `DLPRootMetrics` contract inherits from the `ERC2771ContextUpgradeable` implementation, however this new migration version does not need to make use of any ERC2771 features. While the contract import cannot be removed (the storage layout must be preserved), there are many functions and overrides that can be removed:

```
_msgSender
_msgData
_contextSuffixLength
_checkRole
_trustedForwarder
trustedForwarder
updateTrustedForwarder
```

Recommendation(s): Consider removing the following unnecessary functions and overrides in `DLPRootMetrics`.

Status: Unresolved

Update from the client:

6.4 [Info] Updating epochSize requires an upgrade of DLPRootMetrics

File(s): *.cairo

Description: The function updateEpochSize can be called by admins on the DLPRootEpoch contract to change the amount of days for an epoch. However, changing this value will either cause stake amount adjustments to be calculated incorrectly, or even cause reverts while creating new stakes. This is due to the multiplier array in the DLPRootMetrics contract. This array contains hardcoded values suited for an epoch size of exactly 20 days.

```
function _createStake(address stakerAddress, uint256 dlpId, uint256 amount, uint256 startBlock) internal {
    // ...
    IDLPRootEpoch.EpochInfo memory epoch = dlpRootEpoch.epochs(epochsCount);
    if (startBlock >= epoch.startBlock && epochsCount >= NEW_MULTIPLIER_EPOCH) {
        // @audit The comment below only holds assuming the multiplier matches the epochSize
        // we know that amount > calculateStakeScore(amount, startBlock, _epochs[epochsCount].endBlock
        // because the multiplier during the current epoch is less than 10000
        dlpRootMetrics.updateEpochDlpStakeAmountAdjustment(
            epochsCount,
            dlpId,
            amount - calculateStakeScore(amount, startBlock, dlpRootEpoch.epochs(epochsCount).endBlock),
            true
        );
    }
    // ...
}
```

Consider a case where the epochSize is changed to 30 days, but the multiplier remains configured for 20 days, where at day 20 the user will receive their full stake score. Creating a stake at the beginning of an epoch would result in a calculated stake score that is higher than the amount variable in the code snippet above, since the multiplier is configured for 20 days, so a 30 day stake would give a reward (an additionl amount). This would result in the subtraction causing the transaction to revert.

If the epoch size is updated, the values in the multiplier array should be changed too, to ensure correct stake scoring and rewards. Since this array is hardcoded, this would require a proxy upgrade of the DLPRootMetrics contract to contain the updated array.

Recommendation(s): No changes are necessary, the intention of this finding is to highlight the full process required to change the epoch size, as the relationship between epochSize and multiplier are not obviously linked. Consider adding documentation to explain this behavior.

Status: Unresolved

Update from the client:

6.5 [Best Practices] createEpochs can be used instead of createEpochsWithBlockNumber

File(s): DLPRootImplementation.cairo

Description: The DLPRoot makes calls to the DLPRootEpoch contract in multiple logic flows in order to create new epochs before processing staking or reward related calculations. On the DLPRootEpoch there are two functions that allow new epochs to be created: createEpochs which creates epochs until the current block, and createEpochsUntilBlockNumber which creates epochs up until the provided block number as an argument. The DLPRoot always calls createEpochsUntilBlockNumber but always passes the current block number as an argument. In this case, it would be more appropriate to use createEpochs.

```
// These both have the same result
// CreateEpochs is more gas efficient as no arguments are passed
dlpRootEpoch.createEpochsUntilBlockNumber(block.number);
dlpRootEpoch.createEpochs();
```

Recommendation(s): Consider using createEpochs in the DLPRoot instead of createEpochsUntilBlockNumber.

Status: Unresolved

Update from the client:

7 Documentation Evaluation

Software documentation refers to the written or visual information describing software's functionality, architecture, design, and implementation. 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 Braavos Wallet 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 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 Test Suite Evaluation

9.1 Compilation Output

```
user@machine vana-smart-contracts % npx hardhat compile
Warning: This declaration has the same name as another declaration.
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:443:9:
    |
443 |         address payable foundationWalletAddress
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: The other declaration is here:
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:103:5:
    |
103 |     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:581:9:
    |
581 |         IDLPRootMetrics.DlpRating[] memory topDlps = topDlpsDefaultPercentages(
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:131:5:
    |
131 |     function topDlps(
    |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/rootMetrics/DLPRootMetricsImplementation.sol:482:9:
    |
482 |         bool shouldFinalize,
    |         ^^^^^^^^^^^^^^^^^^^


Generating typings for: 43 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 142 typings!
Compiled 43 Solidity files successfully (evm target: paris).
```

9.2 Tests Output

```
TODO
```

10 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 procehttps://www.overleaf.com/project/65c0e737f41a29601bda5c48ss, 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.