# Security Review Report
# NM-0451-0503 Vana-Data-Access

(April 24, 2025)

# Contents

# 1 Executive Summary

This document presents the security review conducted by Nethermind Security for Vana's Data Access contracts. The audit specifically focused on pull request #14, which introduces the three major components of the Data Access System: data refinement, query engine, and compute engine.

**The audit comprised of** 1403 lines of solidity code and was performed using (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases.

**Along this document, we report** 9 points of attention, where they 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 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.

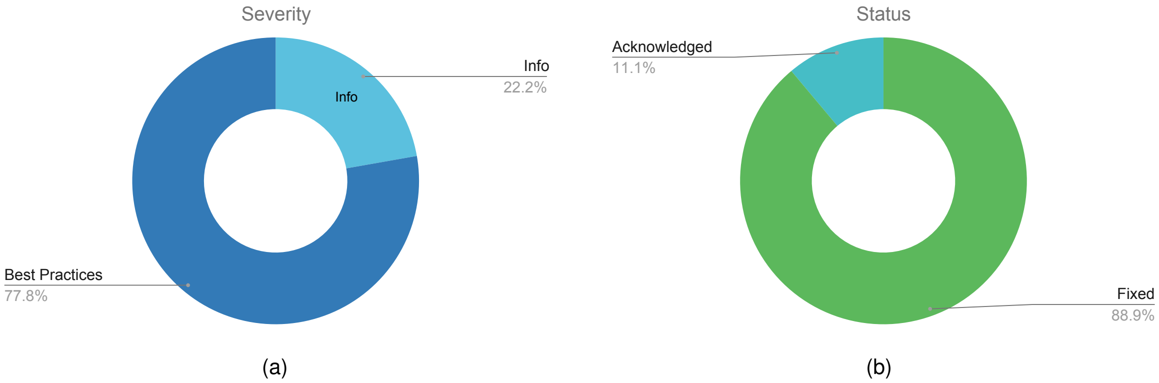


**Fig. 1: Distribution of issues: Critical** (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (2), **Best Practices** (7).
**Distribution of status: Fixed** (8), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

### Summary of the Audit

| | |
|---|---|
| **Audit Type** | Security Review |
| **Initial Report** | April 11, 2025 |
| **Final Report** | April 24, 2025 |
| **Repositories** | vana-smart-contracts |
| **Initial Commit** | b3045f9 |
| **Final Commit** | 967896b |
| **Documentation** | Provided pull request #14 documentation |
| **Documentation Assessment** | Medium |
| **Test Suite Assessment** | Medium |

## 2 Audited Files

| | Files | LoC | Comments | Ratio | Blank | Total |
|---|---|---|---|---|---|---|
| 1 | ComputeEngineImplementation.sol | 341 | 57 | 16% | 74 | 472 |
| 2 | ComputeEngineProxy.sol | 5 | 1 | 20% | 2 | 8 |
| 3 | ComputeEngineTeePoolImplementation.sol | 167 | 35 | 21% | 39 | 241 |
| 4 | ComputeEngineTeePoolProxyFactory.sol | 19 | 1 | 5% | 5 | 25 |
| 5 | ComputeEngineTeePoolFactoryImplementation.sol | 164 | 22 | 13% | 32 | 218 |
| 6 | ComputeInstructionRegistryImplementation.sol | 79 | 22 | 27% | 20 | 121 |
| 7 | ComputeInstructionRegistryProxy.sol | 5 | 1 | 20% | 2 | 8 |
| 8 | DataAccessTreasuryImplementation.sol | 56 | 12 | 21% | 15 | 83 |
| 9 | DataAccessTreasuryProxyFactory.sol | 18 | 1 | 5% | 4 | 23 |
| 10 | DataRefinerRegistryImplementation.sol | 73 | 21 | 28% | 19 | 113 |
| 11 | DataRefinerRegistryProxy.sol | 5 | 1 | 20% | 2 | 8 |
| 12 | DataRegistryImplementation.sol | 180 | 127 | 70% | 50 | 356 |
| 13 | QueryEngineImplementation.sol | 286 | 54 | 18% | 63 | 401 |
| 14 | QueryEngineProxy.sol | 5 | 1 | 20% | 2 | 8 |
| | **Total** | **1403** | **488** | **35** | **329** | **2220** |

## 3 Summary of Issues

| | Finding | Severity | Update |
|---|---|---|---|
| 1 | Improper access control management of Refiner when DLP ownership is transferred | Info | Fixed |
| 2 | Native token can be lost when depositing an ERC20 token | Info | Fixed |
| 3 | Inconsistent casting of `maxTimeout` | Best Practices | Fixed |
| 4 | `owner` address existence check can be optimized to save gas | Best Practices | Fixed |
| 5 | Inaccurate error emission | Best Practices | Fixed |
| 6 | Manual initialization of `TeepoolFactory` could cause transaction revert when attempting bulk update | Best Practices | Fixed |
| 7 | The owner privileges can be permanently lost when updating contract `custodian` role | Best Practices | Fixed |
| 8 | Inconsistent key to value pair in `_jobPayments` mapping | Best Practices | Fixed |
| 9 | Inconsistent code logic with code comments | Best Practices | Acknowledged |

# 4 System Overview

## 4.1 Data Refinement

**Vana's** Data refinement process ascertains the security standards and verifiable quality of the ingested datasets before storage. The refinement process steps include data normalization to ensure the data is structured according to the on-chain schema, masking, which hides any information DLP owners do not want to provide access to, and encryption to prevent unauthorized access. The key contract components in Vana's Data refinement process include the **DataRegistry**, which adds refinements into data files, and the **DataRefinerRegistry**, which manages refiners that can be used to generate data refinements.

```
    struct Refiner {
        uint256 dlpId;
        address owner;
        string name;
        string schemaDefinitionUrl;
        string refinementInstructionUrl;
        string publicKey;

    struct ProofData {
        uint256 score;
        uint256 dlpId;
        string metadata;
        string proofUrl;
        string instruction;
    }
```

## 4.2 Query Engine

**Vana's** Query Engine handles query payments for data access and manages access and permissions to refined data. The Query Engine is only accessible through the Compute Engine via Compute Engine Jobs. Data requestors submit registered jobs to the Compute Engine with a query scoped to the authorized dataset. The Query Engine access control structure ensures only authorized data requestors can query refined data.

```
     struct Permission {
        address grantee;
        bool approved;
        uint256 refinerId;
        string tableName;
        string columnName;
        uint256 price;
    }

    struct PermissionInfo {
        uint256 permissionId;
        address grantee;
        bool approved;
        uint256 refinerId;
        string tableName;
        string columnName;
        uint256 price;
    }
```

## 4.3 Compute Engine

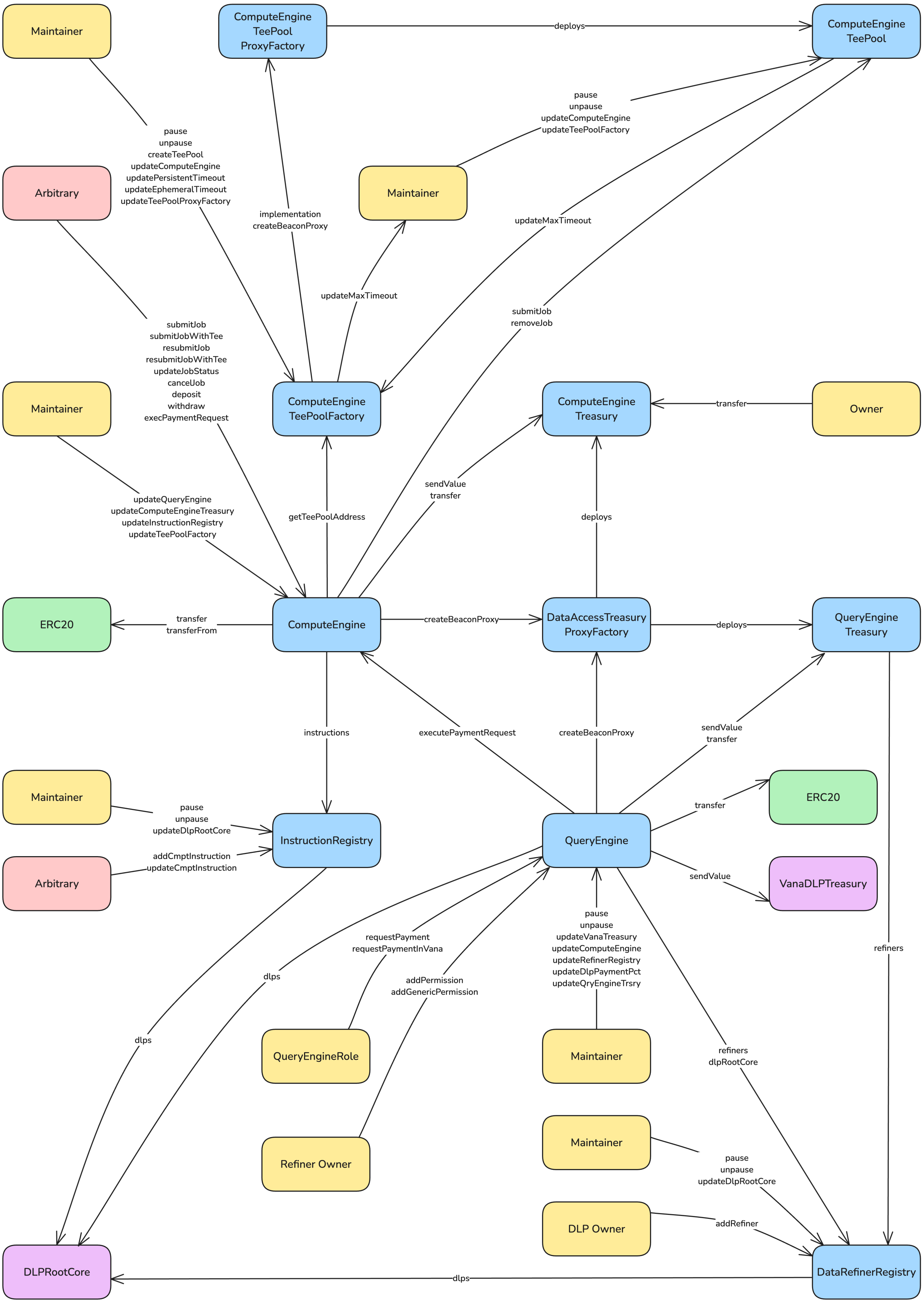
**Vana's** Compute Engine ensures safe operations on encrypted data through running containerized jobs and acts as a gateway to the Query Engine. The Compute Engine jobs are designed to be reusable and produce artifacts that are available for a defined period before expiration.

Additionally, the Compute Engine handles user deposits for data access and processes payment requests from the Query Engine.

```
struct PaymentInfo {
        address payer;
        mapping(address token => uint256 amount) paidAmounts;
    }
```

## 4.4 Vana Data Access Protocol Diagram

The below diagram highlights various contract components and their interactions in Vana's Data Access Module.


Maintainer
ComputeEngine TeePool ProxyFactory
deploys
ComputeEngine TeePool
pause
unpause
updateComputeEngine
updateTeePoolFactory
pause
unpause
createTeePool
updateComputeEngine
updatePersistentTimeout
updateEphemeralTimeout
updateTeePoolProxyFactory
Arbitrary
implementation
createBeaconProxy
Maintainer
updateMaxTimeout
updateMaxTimeout
submitJob
submitJobWithTee
resubmitJob
resubmitJobWithTee
updateJobStatus
cancelJob
deposit
withdraw
execPaymentRequest
submitJob
removeJob
Maintainer
ComputeEngine TeePoolFactory
ComputeEngine Treasury
transfer
Owner
sendValue
transfer
updateQueryEngine
updateComputeEngineTreasury
updateInstructionRegistry
updateTeePoolFactory
getTeePoolAddress
deploys
ERC20
transfer
transferFrom
ComputeEngine
createBeaconProxy
DataAccessTreasury ProxyFactory
deploys
QueryEngine Treasury
instructions
executePaymentRequest
createBeaconProxy
sendValue
transfer
Maintainer
pause
unpause
updateDlpRootCore
InstructionRegistry
QueryEngine
transfer
ERC20
Arbitrary
addCmptInstruction
updateCmptInstruction
sendValue
VanaDLPTreasury
requestPayment
requestPaymentInVana
pause
unpause
updateVanaTreasury
updateComputeEngine
updateRefinerRegistry
updateDlpPaymentPct
updateQryEngineTrsry
refiners
dlps
addPermission
addGenericPermission
dlps
QueryEngineRole
Maintainer
refiners
dlpRootCore
Refiner Owner
Maintainer
pause
unpause
updateDlpRootCore
DLP Owner
addRefiner
DLPRootCore
dlps
DataRefinerRegistry

# 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 [Info] Improper access control management of `Refiner` when DLP Ownership is transferred

**File(s)**: `DataRefinerRegistryImplementation.sol`

**Description**: In `addRefiner(...)` function, the `owner` is set to the current owner of the `dlpId`, but if the owner is transferred then `Refiner` will still correspond to the old owner. This is an issue given functions `addPermission(...)` and `updatePermissionApproval` from the Query Engine uses the owner field of the `Refiner` struct.

**Recommendation(s)**: Consider updating `Refiner` ownership whenever there is a DLP ownership transfer. This behavior should also be properly documented and communicated to DLP owners.

**Status**: Fixed

**Update from the client**: Fixed in commit 967896b

## 6.2 [Info] Native token can be lost when depositing an ERC20 token

**File(s)**: `ComputeEngineImplementation.sol`

**Description**: In the `deposit(...)` function of the `ComputeEngineImplementation` contract, it is possible to deposit with a nonzero **msg.value** given the function is marked payable while using an ERC20 token. In that case, the contract accepts native asset funds, but the deposit will be registered under the token. As a result, the function will attempt to transfer that token from the user to itself and then to the treasury, which would be double paying. The native asset is not tracked in this case and can be lost.

**Recommendation(s)**: Consider adding a check if the token is VANA, then the `msg.value` must be zero.

**Status**: Fixed

**Update from the client**: Fixed in commit a8cc24d

## 6.3 [Best Practices] Inconsistent casting of `maxTimeout`

**File(s)**: `ComputeEngineImplementation.sol`

**Description**: In the internal `_registerJob (...)` the passed parameter `maxTimeout` is of type `uint256`. However, in functions `submitJob(...)` and `submitJobWithTee(...)` where `_registerJob(...)` is called, the `maxTimeout` is passed as a `uint80` data type. For consistency, the project should consider changing the `maxTimeout` type in `_registerJob(...)` to a `uint80` and avoid silent casting.

**Recommendation(s)**: Consider skipping the iteration for duplicate accounts to avoid double counting.

**Status**: Fixed

**Update from the client**: Fixed in commit a8cc24d

## 6.4 [Best Practices] Owner address existence check can be optimized to save gas

**File(s)**: `ComputeEngineImplementation.sol`

**Description**: In `_registerJob()` function, the `instructions` registry is queried just to check the owner's address as seen below:

```
function _registerJob(
        uint256 maxTimeout,
        bool gpuRequired,
        uint256 computeInstructionId
    ) internal whenNotPaused returns (uint256 jobId) {
        IComputeInstructionRegistry.ComputeInstructionInfo memory computeInstruction = instructionRegistry.instructions(
            computeInstructionId
        );
        ........
        .......
```

This can be optimized further by adding a simple existence check directly in the instruction registry.

**Recommendation(s)**: Consider directly adding a simple existence check in the instruction registry to optimize the code and save gas.

**Status**: Fixed

**Update from the client**: Fixed in commit a8cc24d

## 6.5 [Best Practices] Inaccurate error emission

**File(s)**: ComputeEngineImplementation.sol

**Description**: When updating job status via updateJobStatus(...) function, the contract checks the current job status if it is completed or Failed. If this is true, JobAlreadyDoneOrCanceled() error is emitted. The same inconsistency exists in cancelJob(...) function where the error of canceling a job is JobAlreadyDoneOrCanceled() but the code logic covers done, canceled and failed cases. Similarly, in _resubmitJobWithTee (...) the error message JobAlreadyDoneOrCanceled() would be incorrect if job.status is Submitted.

**Recommendation(s)**: Consider ensuring consistent error emission with code logic.

**Status**: Fixed

**Update from the client**: Fixed in commit a8cc24d

## 6.6 [Best Practices] Manual Initialization of TeePoolFactory could cause transaction revert when attempting bulk update

**File(s)**: ComputeEngineTeePoolFactoryImplementation.sol

**Description**: After the TeePool is deployed via the beacon proxy, the user with maintainer role must manually set the teepoolFactory address without which, maxTimeout updates would fail. This could cause reverts when attempting to bulk update.

```
function updateTeePoolProxyFactory(
        ComputeEngineTeePoolProxyFactory newTeePoolProxyFactory
    ) external override onlyRole(MAINTAINER_ROLE) {
        teePoolProxyFactory = newTeePoolProxyFactory;
    }
```

**Recommendation(s)**: Consider setting the teepoolFactory address when initializing the contract.

**Status**: Fixed

**Update from the client**: Fixed in commit a8cc24d

## 6.7 [Best Practices] The owner privileges can be permanently lost when updating contract custodian role

**File(s)**: DataAccessTreasuryImplementation.sol

**Description**: When updating the contract custodian, there is no check to ensure the newCustodian is not the owner's address.

```
function updateCustodian(address newCustodian) external override onlyRole(DEFAULT_ADMIN_ROLE) {
        _revokeRole(DEFAULT_ADMIN_ROLE, custodian);
        custodian = newCustodian;
        _grantRole(DEFAULT_ADMIN_ROLE, newCustodian);
    }
```

Without this check, it is possible to set the custodian as the owner's address, effectively leaving the contract with only one custodian. If the custodian is changed again, the owner could lose privileges permanently.

**Recommendation(s)**: Consider adding a check preventing the contract owner from becoming the contract's custodian.

**Status**: Fixed

**Update from the client**: Fixed in commit a8cc24d

## 6.8 [Best Practices] Inconsistent key to value pair in `_jobPayments` mapping

**File(s)**: `ComputeEngineStorageV1.sol`

**Description**: In `ComputeEngineStorageV1` contract, the mapping `_jobPayments` has a mapping key `provider` seen as `mapping(uint256 jobId => mapping(address provider => PaymentInfo paymentInfo)) internal _jobPayments;`. However, the implementation logic in `_executePaymentRequestFromQueryEngine (...)` uses the token address as the key for this slot and then a token address again within a mapping from the resulting struct.

```
function _executePaymentRequestFromQueryEngine(address token, uint256 amount, bytes calldata metadata) internal {
       ...........
       ...........
        PaymentInfo storage paymentInfo = _jobPayments[jobId][token];
        paymentInfo.paidAmounts[token] += amount;
        paymentInfo.payer = jobOwner;

        emit PaymentExecuted(jobId, token, amount);

        computeEngineTreasury.transfer(msg.sender, token, amount);
    }
```

A token is not a provider. This inconsistency should be addressed by changing the provider to another mapping key.

**Recommendation(s)**: Consider updating the provider to another `key` in the mapping.

**Status**: Fixed

**Update from the client**: Fixed in commit a8cc24d

## 6.9 [Best Practices] Inconsistent code logic with code comments

**File(s)**: `DataRegistryImplementation.sol`

**Description**: In `DataRegistryImplementation` contract, the code comments within `addRefinementWithPermission(...)` permission states: `"The permission for an account is not allowed to be changed once set"`.

```
    function addRefinementWithPermission(
        uint256 fileId,
        uint256 refinerId,
        string calldata url,
        address account,
        string calldata key) external override whenNotPaused onlyRole(REFINEMENT_SERVICE_ROLE) {

       ...........
       ...........
            // @dev Add permission for the account to access the refinement.
        // The permission for an account is not allowed to be changed once set,
        // to prevent previous refinements from being inaccessible.
        if (bytes(_file.permissions[account]).length == 0) {
            _file.permissions[account] = key;
            emit PermissionGranted(fileId, account);
        }
    }
```

However, the `addFilePermission (...)` function allows the account's permission to be updated.

**Recommendation(s)**: Consider either updating the code logic, reflecting the code comments, or updating the code comments to reflect the code logic.

**Status**: Acknowledged

**Update from the client**: This is 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 Vana Data Access documentation

The `Vana` team has provided a comprehensive walkthrough of the project as well as documentation highlighting the pull request changes. Moreover, the team addressed all questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

# 8 Test Suite Evaluation

## 8.1 Compilation Output

```
> npx hardhat compile
Generating typings for: 172 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 502 typings!
Compiled 167 Solidity files successfully (evm target: paris).
```

## 8.2 Tests Output

```
npx hardhat test
ComputeEngine
    Setup
       should have correct params after deploy
       should have correct treasury address after deploy
       should grant or revoke roles when admin
       should upgradeTo when owner
       should not upgradeTo when non-owner
       should upgradeTo when owner and emit event
       should reject upgradeTo when storage layout is incompatible
       should not initialize in implementation contract
       should pause and unpause only when maintainer
       should updateQueryEngine only when maintainer
       should updateTeePoolFactory only when maintainer
       should updateInstructionRegistry only when maintainer
    TeePool Factory
       should createTeePool only when maintainer
       should not createTeePool when invalid TeePoolType
       should not create duplicate TeePool
       should updateEphemeralTimeout only when maintainer
       should not updateEphemeralTimeout when invalid timeout
       should updatePersistentTimeout only when maintainer
       should not updatePersistentTimeout when invalid timeout
    TeePool
       should addTee only when maintainer
       should removeTee only when maintainer
       should submitJob to active Tees only
    Job Registry
       should registerJob without TeePool
       should registerJob with empty TeePool
       should not submitJob when invalid computeInstructionId
       should submitJob with non-empty TeePool
       should resubmitJob when Tee is available
       should submitJobWithTee when Tee is available
       should resubmitJobWithTee when the dedicated Tee is available
       should updateJobStatus only when assigned Tee
       should submitJob to correct TeePool and Tee
       should cancelJob when owner
       should not submitJob/submitJobWithTee/resubmitJob/cancelJob/updateJobStatus when paused
    Payment
       should deposit and withdraw VANA
       should deposit and withdraw ERC20
       should executePaymentRequest from queryEngine
       should nonReentrant
       should not deposit/withdraw/executePaymentRequest when paused
  DataRefinerRegistry
    Setup
       should have correct params after deploy
       should grant or revoke roles when admin
       should upgradeTo when owner
       should not upgradeTo when non owner
       should upgradeTo when owner and emit event
       should reject upgradeTo when storage layout is incompatible
       should pause and unpause only when maintainer
       should updateDlpRootCore only when maintainer
    addRefiner
       should addRefiner only when DLP owner
       should not addRefiner when pause

  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
    AddRefinementWithPermission
       should addRefinementWithPermission
       should addRefinementWithPermission against multiple refiners
       should not addRefinementWithPermission with invalid fileId
       should not allow unauthorized users to addRefinementWithPermission
       should not addRefinementWithPermission with empty URL
       should not addRefinementWithPermission more than once against the same refiner

  QueryEngine
    Setup
       should have correct params after deploy
       should have correct treasury addresses after deploy
       should grant or revoke roles when admin
       should upgradeTo when owner
       should not upgradeTo when non-owner
       should upgradeTo when owner and emit event
       should reject upgradeTo when storage layout is incompatible
       should not initialize in implementation contract
       should pause and unpause only when maintainer
       should updateRefinerRegistry only when maintainer
       should updateComputeEngine only when maintainer
       should updateQueryEngineTreasury only when maintainer
       should updateVanaTreasury only when maintainer
    Permissions
       should addPermission only when DLP owner
       should updatePermissionApproval only when DLP owner
       should not allow non-empty columnName when tableName is empty
    Payments
       should requestPaymentInVana when queryEngineTEE
       should revert when not QUERY_ENGINE_ROLE
       should revert when jobId is not found
       should revert when refinerId is not found
       should revert when token is not VanaToken or ERC20
       should revert when user balance is insufficient
       should revert when the payment is not received
       should revert when reentrancy
       should revert when non-DlpOwner claimDlpPayment
       should revert when dlpTreasuryAddress is not set
       should not claimDlpPayment when paused
       should not claimDlpPayment when queryEngineTreasury paused
       should not requestPaymentInVana when computeEngineTreasury paused
```

## 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.