# Security Review Report
# NM-0650 and NM-0675 - Ventuals


(October 8, 2025)

# Contents

# 1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for **Ventuals** LST contracts.

**Ventuals** brings the power of perps to the most innovative private companies in the world, which is a multi-trillion dollar asset class that normal investors have historically been locked out of.

Ventuals is built on Hyperliquid's proven perps orderbook infrastructure, using the HIP-3 standard.

Ventuals created its own HYPE LST (vHYPE) to raise the HIP-3 stake requirement. vHYPE is a fully transferable ERC20 token, and serves as the claim to the original HYPE plus accrued native staking yield.

The security review was performed in two phases: an initial assessment of the Genesis contracts and a subsequent review of the version adapted for HIP-3.

The initial scope covered the Genesis phase contracts, which allowed users to stake HYPE tokens for a one-year lock-up. In exchange, users received vHYPE, a transferable ERC20 token representing their staked position. The protocol featured auto-compounding for all native staking rewards and was designed for future enhancements using a UUPS upgradeable proxy pattern.

The second phase of the review focused on significant updates, headlined by the introduction of the StakingVaultManager contract to replace the GenesisVaultManager. This new contract introduces a batched withdrawal system with a 7-day delay queue. Furthermore, a slashing mechanism was implemented that can affect both actively staked funds and assets in the withdrawal queue. These changes required corresponding modifications to the RoleRegistry, StakingVault, and VHYPE contracts.

**The audit comprises 1000** lines of Solidity code. **The audit was performed using** (a) manual analysis of the codebase, and (b) automated analysis tools.

**Along this document, we report** 24 points of attention where one is classified as `Critical`, two are classified as `high`, eight are classified as `Medium`, five are classified as `Low` and eight are classified as `Informational and Best Practices` severity. 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 and Section 7 provide details of the issues from intial and the new version. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the test suite evaluation and automated tools used. Section 10 concludes the document.

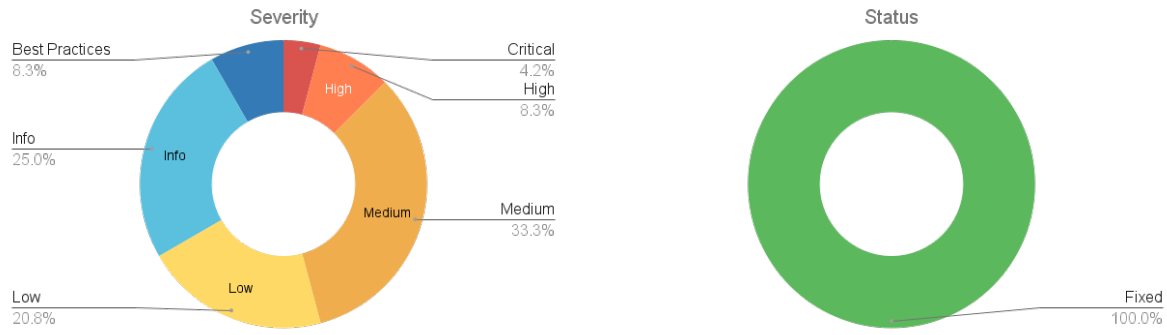


**Fig. 1: Distribution of issues: Critical** (1), **High** (2), **Medium** (8), **Low** (5), **Undetermined** (0), **Informational** (6), **Best Practices** (2). **Distribution of status: Fixed** (24), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

### Summary of the Audit

| | |
|---|---|
| **Audit Type** | Security Review |
| **Final Report** | September 23, 2025 |
| **New Version Final Report** | October 8, 2025 |
| **Initial Commit** | 2e57c08c0fd6fb898d6f28670d6bb985398f6daa |
| **Final Commit** | d21f1a94026fd7cb790b39449d45e4c11f93111a |
| **New Version Initial Commit** | 024ff6ef3d0272a54c14808c770eb3b2864123ba |
| **New Version Final Commit** | 522f247c7ce041f21df9cf28dc05b6466da83124 |
| **Documentation Assessment** | High |
| **Test Suite Assessment** | High |

**Remarks about Hyperliquid documentation**

The second part of the review was conducted according to the changes introduced on the `Hyperliquid` platform in HIP-3. However, Hyperliquid did not provide a detailed explanation of the slashing event. In particular, it is unclear whether there is any lock-up period for funds after slashing. The Ventuals protocol introduced a pausing mechanism that can be used during slashing to mitigate potential negative side effects, which were not documented. Therefore, we strongly recommend contacting Hyperliquid for clarification and implementing the necessary security measures.

# 2 Audited Files

| | Contract | LoC | Comments | Ratio | Blank | Total |
|---|---|---|---|---|---|---|
| 1 | GenesisVaultManager.sol | 170 | 129 | 75.9% | 62 | 361 |
| 2 | RoleRegistry.sol | 41 | 22 | 53.7% | 14 | 77 |
| 3 | Base.sol | 40 | 6 | 15.0% | 10 | 56 |
| 4 | StakingVault.sol | 55 | 15 | 27.3% | 18 | 88 |
| 5 | VHYPE.sol | 15 | 2 | 13.3% | 4 | 21 |
| 6 | interfaces/IStakingVault.sol | 17 | 28 | 164.7% | 14 | 59 |
| 7 | interfaces/ICoreWriter.sol | 4 | 1 | 25.0% | 1 | 6 |
| 8 | libraries/CoreWriterLibrary.sol | 42 | 24 | 57.1% | 10 | 76 |
| 9 | libraries/Converters.sol | 10 | 3 | 30.0% | 3 | 16 |
| 10 | libraries/L1ReadLibrary.sol | 218 | 3 | 1.4% | 32 | 253 |
| | **Total** | **612** | **233** | **38.1%** | **168** | **1013** |

The new version includes file `StakingVaultManager.sol` which was introduced in the place of `GenesisVaultManager.sol`.

| | Contract | LoC | Comments | Ratio | Blank | Total |
|---|---|---|---|---|---|---|
| 1 | StakingVaultManager.sol | 388 | 236 | 60.8% | 157 | 781 |
| | **Total** | **388** | **236** | **60.8%** | **157** | **781** |

Additionally, the new version introduced the following number of line changes to existing files:

| | Contract | Changed LoC |
|---|---|---|
| 1 | RoleRegistry.sol | 4 |
| 2 | StakingVault.sol | 82 |
| 3 | VHYPE.sol | 11 |

# 3 Summary of Issues

| | Finding | Severity | Update |
|---|---|---|---|
| 1 | Exchange Rate Manipulation Leading to Inflation Attack | High | Fixed |
| 2 | Not burning vHype on transfer | High | Fixed |
| 3 | Decimal Conversion Loss for Deposited Amounts | Medium | Fixed |
| 4 | Manager role to be restricted to GenesisVaultManager | Medium | Fixed |
| 5 | Hype tokens deposited directly into GenesisVault Manager are stuck | Low | Fixed |
| 6 | Malicious user can inflate the price of vHype | Low | Fixed |
| 7 | _mint() can be called when the contract is paused | Low | Fixed |
| 8 | Event log can be cluttered with 0 deposits | Info | Fixed |
| 9 | Inconsistent comment for weiAmount conversion in CoreWriterLibrary | Info | Fixed |
| 10 | emergencyStakingWithdraw may fail | Info | Fixed |
| 11 | whitelistDepositLimits should be greater than current defaultDepositLimit | Info | Fixed |
| 12 | Upgradeable contract lacks storage gaps | Best Practices | Fixed |

Issues identified in the new version:

| | Finding | Severity | Update |
|---|---|---|---|
| 1 | Withdrawals can be claimed before finalisation | Critical | Fixed |
| 2 | Lack of pause for `burnFrom()` | Medium | Fixed |
| 3 | Lack of withdrawal request minimum amount can lead to DOS | Medium | Fixed |
| 4 | Missing Validator address validation | Medium | Fixed |
| 5 | No limit on withdrawal request | Medium | Fixed |
| 6 | Problems with applySlash() after claiming withdrawals | Medium | Fixed |
| 7 | Truncation during HyperEVM to HyperCore transfer | Medium | Fixed |
| 8 | Unclear error message when claiming unprocessed withdraw | Low | Fixed |
| 9 | Validation logic should be moved to StakingVault | Low | Fixed |
| 10 | Unclear timing logic in delegation reading | Info | Fixed |
| 11 | minimumStakeBalance changes can temporarily block batch finalization | Info | Fixed |
| 12 | Increase timing checks | Best Practices | Fixed |

# 4 Protocol Overview

The *Ventuals* LST protocol is a multi-contract system designed to operate across two distinct layers of the *Hyperliquid network*.

- *HyperEVM(L2)*: This layer hosts the smart contracts
- *HyperCore(L1)*: This layer handles the native *HYPE* staking with the validators.

The main data flow involves *HYPE* tokens moving from a user's wallet on *HyperEVM* to the *StakingVault* contract, then being bridged to *HyperCore* where they are delegated to validators. Staking rewards accumulated on *HyperCore*, increasing the total *HYPE* held by the vault, which in turn appreciates the value of the *vHYPE* token.

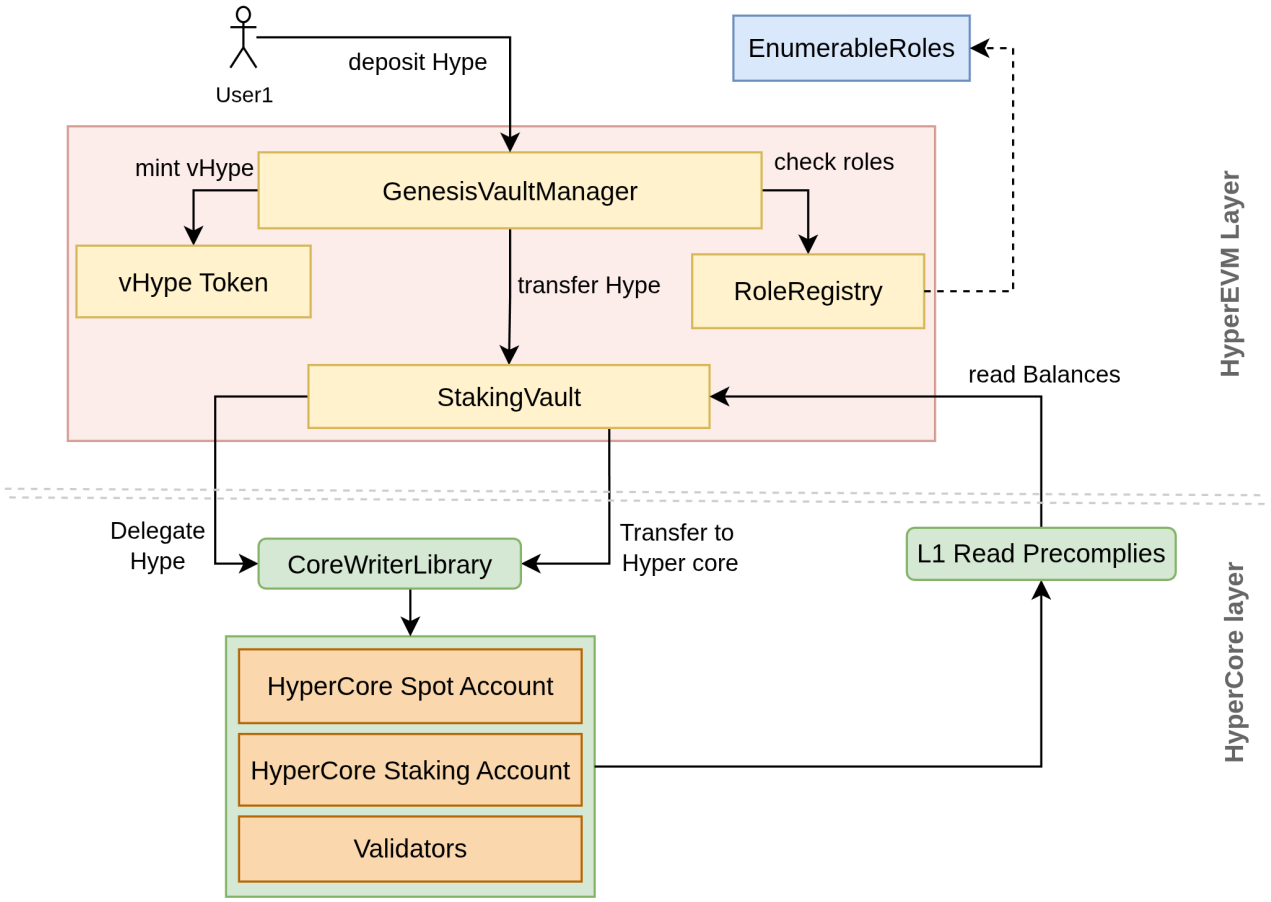


**Fig. 2: Ventuals LST overview**

## 4.1 Core Components

Below is an overview of smart contracts and their functionality in the protocol.

### 4.1.1 GenesisVaultManager

a) **Purpose**: Is the primary entry point for users to stake *Hype* tokens. It manages key state variables related to the capacity of the protocol, deposits at the user level, and the exchange rate for *Hype* to *vHype* conversion.

b) **State**:

- *totalVaultCapacity*: The maximum amount of *Hype* the protocol can accept.
- *individualDepositLimit*: The maximum *Hype* a single user can deposit.
- *currentExchangeRate*: The calculated ratio of *HYPE* to *vHYPE*. This is a dynamic value derived from the total *HYPE* in the system relative to the total supply of *vHYPE*.

c) **Functionality**:

- *deposit*: The main public function to accept *Hype* deposits from users.
- It interacts with *StakingVault* to receive the deposited *HYPE* and with the *vHYPE* contract to mint the new tokens.

#### 4.1.2 vHYPE

a) **Purpose**: A standard ERC20 implementation with additional logic for minting and burning. It represents a proportional share of the staked *HYPE*.

b) **Mechanism of Value Appreciation**: As staking rewards on *HyperCore* increase the *HYPE* balance in the *StakingVault*, the numerator of the ratio increases while the denominator (vHYPE supply) remains constant, thus increasing the value of each *vHYPE* token.

c) **Minting**: The *mint* function is permissioned and can only be called by the *GenesisVaultManager* or a privileged Manager role.

#### 4.1.3 StakingVault

a) **Purpose**: A specialized contract that acts as the secure holder of deposited *HYPE* and the main communication layer between *HypeEVM* L2 and *HyperCore*'s L1 precompiled contracts.

b) **L1-L2 Bridge**: It utilizes HyperEVM's precompiled contracts to execute actions on HyperCore. This is a critical architectural detail that enables cross-layer communication without complex bridging protocols.

c) **Key Functionality**:

- *transferHypeToCore*: Transfers *HYPE* from the vault's *HyperEVM* balance to its associated spot account on *HyperCore*.
- *stakingDeposit*: Uses a *HyperCore* precompiled function to move *HYPE* from the spot account to a staking account on L1.
- *tokenDelegate* : A key function that interacts directly with *HyperCore*'s precompiled contracts to delegate or undelegate *HYPE* to a specific validator. This is how staking is initiated and managed.
- *stakingWithdraw*: Initiates the withdrawal process from the staking account on *HyperCore*.

#### 4.1.4 RoleRegistry

a) **Purpose**: A centralized, modular access control system based on an address => role mapping. Isolates the access control logic from the business logic.

b) **Roles and Permissions**:

- Owner: The highest level of control. Can manage roles, and pause/unpause contracts.
- Manager: A highly privileged role with direct control over critical vault functions such as minting *vHYPE* and initiating staking/unstaking processes on *HyperCore*.
- Operator: A more limited role, typically for automated or off-chain agents. Their primary function is to call transferToCoreAndDelegate to move HYPE from the vault to the L1 staking system.

c) **Mechanism**: Functions in other contracts (e.g., StakingVault) use modifiers like onlyManager or onlyOperator to ensure that only addresses with the correct role can call them, as defined by the RoleRegistry.

### 4.2 Interactions and Data Flow

#### 4.2.1 User Deposit

- A user calls *GenesisVaultManager*'s *deposit* function.
- *GenesisVaultManager* performs a token transfer of *HYPE* from the user to the *StakingVault* contract.
- *mints* vHYPE based on the current exchange rate and sends them to the user.

#### 4.2.2 HYPE Delegation

- *transferToCoreAndDelegate* function moves the HYPE from the L2 contract balance to the L1 spot account.
- calls *stakingDeposit* to move the tokens to the L1 staking account.
- Finally, it calls *tokenDelegate* to delegate the *HYPE* to a validator on *HyperCore*.

#### 4.2.3 Reward Accumulation

- Staking rewards are generated on the *HyperCore* L1 and automatically increase the *HYPE* balance in the *StakingVault*'s staking account.
- This balance change is not a direct L2 state update but is reflected in the total *HYPE* when *GenesisVaultManager* calculates the exchange rate, thereby increasing the value of *vHYPE* for all holders.

## 4.3 Smart Contracts in New Version

Since the initial audit, the *Ventuals* LST protocol has undergone significant architectural changes that transform the system capabilities and user experience. The most notable change is the complete replacement of the *GenesisVaultManager* with a new *StakingVaultManager* contract, which introduces a batch-based withdrawal system and removes several previous limitations.

### 4.3.1 StakingVaultManager (Replaces GenesisVaultManager)

- **Role**: Serves as the primary entry point for users to stake *HYPE* tokens, replacing the previous *GenesisVaultManager*.
- **Key Additions**: Introduces a batch-based withdrawal mechanism while maintaining the core deposit functionality.
- **Simplifications**: Removes complex capacity management and individual deposit limits; users can deposit *HYPE* without constraints.
- **State Tracking**: Manages withdrawal batches, their processing states, and reserved *HYPE* amounts for pending withdrawals for a more robust and user-friendly experience.

### 4.3.2 New Withdrawal Architecture

- **Batching Model**:
  - Queues user withdrawal requests.
  - Processes them in batches based on available capacity.
  - Imposes a **7-day** waiting period between batch finalization and claimability.
  - Supports slashing events via retroactive exchange-rate adjustments.
- **Dynamic Capacity**: Calculates available withdrawal capacity from the current total balance minus the minimum stake balance and already processed withdrawals.
- **Enhanced Emergency Controls (Owner)**:
  - Pause batch processing.
  - Reset batches prior to finalization.
  - Apply slashing events to specific batches.
  - Execute emergency staking withdrawals and deposits.

### 4.3.3 Updated Flow

**User Deposit (Simplified)**

- User calls `StakingVaultManager.deposit()` with *HYPE*.
- Contract mints *vHYPE* based on the current exchange rate.
- *HYPE* is transferred to the *StakingVault* contract.
- No capacity limits or individual deposit limits are enforced.

**User Withdrawal (New Process)**

- User calls `queueWithdraw()` with the desired *vHYPE* amount.
- *vHYPE* is transferred from the user to the *StakingVaultManager* contract.
- Withdrawal request is added to the withdrawal queue.
- Operator processes withdrawal batches using `processBatch()`.
- Batches are finalized using `finalizeBatch()`, which nets deposits and withdrawals.
- After **7 days**, users can claim their withdrawals using `claimWithdraw()`.

**HYPE Delegation (Enhanced)**

- The system preserves the core delegation mechanism via the *StakingVault*.
- Delegation now occurs as part of the batch finalization process.
- The system automatically stakes excess deposits and unstakes amounts required for withdrawals.

# 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 [High] Exchange Rate Manipulation Leading to Inflation Attack

**File(s)**: src/GenesisVaultManager.sol, src/StakingVault.sol

**Description**:

The HYPETovHYPE function in the GenesisVaultManager contract is vulnerable to an inflation attack due to exchange rate manipulation. The exchange rate is calculated as totalHYPE / totalVHYPE, which can be artificially inflated by donating directly to the vault without minting corresponding shares.

An attacker can exploit this by: (1) depositing a minimal amount (1 wei) to receive 1 vHYPE share, (2) donating a large amount (e.g., 10 ETH) directly to the staking vault via the receive() function, (3) inflating the exchange rate to approximately (10 ETH * 1e18) / 1 wei  10^22, and (4) causing victim deposits to result in 0 vHYPE shares due to integer division truncation when hypeAmount * 1e18 < _exchangeRate in the Math.mulDiv(hypeAmount, 1e18, _exchangeRate) calculation.

This allows the attacker to gain control of all vault assets with minimal investment while legitimate depositors receive 0 vHYPE shares despite depositing significant amounts, resulting in a complete loss of funds for victims and making the vault unusable for legitimate users.

**Recommendation(s)**: Use internal accounting to keep track of all the HYPE in the vault with a storage variable and update it when the contract receives HYPE instead of using .balance.

**Status**: Fixed

**Update from the client**: Fixed by 9007e61d16 and f5da5068c2

## 6.2 [High] Not burning vHype on transfer

**File(s)**: src/GenesisVaultManager.sol, src/StakingVault.sol

**Description**: On depositing Hype tokens into the GenesisVaultManager contract, the tokens are moved into the StakingVault. At this time, the vHype tokens are minted for the accepted amount.

From the StakingVault, the manager can transfer Hype to any other account by calling the transferHype function, which will move the funds out of the protocol.

While Hype tokens are taken out of the protocol, the extent of vHype is not burnt.

While computing the exchange rate, the totalBalance for Hype comprises staking Balance + Spot Balance + funds in the StakingVault contract.

```
function exchangeRate() public view returns (uint256) {
==>  uint256 balance = totalBalance();
==>  uint256 totalSupply = vHYPE.totalSupply();
   //....
   return Math.mulDiv(balance, 1e18, totalSupply);
}
```

As the vHype was not burnt when taking out funds from StakingVault, it will directly impact the exchange rate.

**Recommendation(s)**: Consider removing the transferHype function or burn equivalent vHYPE shares on transfer.

**Status**: Fixed

**Update from the client**: Removed the transferHype function a02ad2aa62

## 6.3 [Medium] Decimal Conversion Loss for Deposited Amounts

**File(s)**: src/GenesisVaultManager.sol

**Description**:

The system suffers from decimal conversion loss when handling HYPE deposits due to the conversion between 18 decimals (EVM standard) and 8 decimals (HyperCore standard).

The `Converters.to8Decimals()` function performs a division by `1e10` to convert from 18 decimals to 8 decimals: This division operation truncates any remainder less than `1e10` wei, resulting in permanent loss of precision. For example:

- A deposit of `1.0000000001` HYPE (1e18 + 100 wei) would be converted to `1` HYPE (1e8) on HyperCore;
- The 100 wei remainder is permanently lost;
- When converted back to 18 decimals, it becomes `1.0000000000` HYPE (1e18), losing the original 100 wei;

**Recommendation(s)**:

Enforce minimum deposit granularity by a validation in the `deposit()` function to ensure only multiples of the minimum precision unit can be deposited (amountToDeposit % 1e10 == 0). Besides, add a minimum deposit amount check to prevent dust deposits that would result in significant relative precision loss.

**Status**: Fixed

**Update from the client:** Added minimum deposit constraint 9007e61d1

## 6.4 [Medium] Manager role to be restricted to GenesisVaultManager

**File(s)**: src/GenesisVaultManager.sol, src/StakingVault.sol

**Description**: In the `StakingVault` there are functions that interact with HyperCore. For the actions performed on `HyperCore`, the updated state does not reflect during reads via pre-compiled contracts until the next block. Hence, any read before the formation of the next block could result in incorrect values.

Currently, in order to prevent such a read, the tracking variable for the last action is in the `GenesisVaultManager` contract. If the manager role is configured for `GenesisVaultManager` and another `EOA` account, there is a possibility for EOA to interact with HyperCore in the `StakingVault`, resulting in changes to HyperCore state in transit.

For example, considering HYPE can be deposited directly into the `StakingVault`, if the intent is to send those funds to any of the below actions interacting with HyperCore, it will impact the working of the protocol.

```
function tokenDelegate(address validator, uint64 weiAmount, bool isUndelegate) external onlyManager whenNotPaused {
    CoreWriterLibrary.tokenDelegate(validator, weiAmount, isUndelegate);
}


/// @inheritdoc IStakingVault
function spotSend(address destination, uint64 token, uint64 weiAmount) external onlyManager whenNotPaused {
    CoreWriterLibrary.spotSend(destination, token, weiAmount);
}


/// @inheritdoc IStakingVault
function transferHypeToCore(uint256 amount) external onlyManager whenNotPaused {
    _transfer(payable(HYPE_SYSTEM_ADDRESS), amount);
}


```

As the protection implemented in `GenesisVaultManager` is not effective in `StakingVault`, it could result in incorrect exchange rates.

**Recommendation(s)**: Consider enforcing the Manager role to `GenesisVaultManager` during the genesis phase which is the intent. An improvement to consider is to move the `_transferToCoreAndDelegate` internal function into `StakingVault` so that interactions between `stakingVault` and `HyperCore` will be effective even after the genesis phase.

**Status**: Fixed

**Update from the client**: As discussed, moved the `lastEvmToCoreTransferBlockNumber` logic to `StakingVault` so that the one-block delay is enforced in the `StakingVault`. 6c4f718eff

## 6.5 [Low] Hype tokens deposited directly into GenesisVault Manager are stuck

**File(s)**: src/GenesisVaultManager.sol

**Description**: GenesisVaultManager can accept Hype tokens directly as the contract implements receive and fallback functions. While the Hype accepted via deposit function, equivalent vHype is minted and tokens are moved to the stakingVault, the directly deposited will remain in the GenesisVaultManager contract.

As there is no function to withdraw the hype tokens from GenesisVaultManager contract, any deposits directly sent to the contract will be stuck and locked forever.

**Recommendation(s)**: Consider removing the receive and fallback functions

**Status**: Fixed

**Update from the client**: Removed the receive and fallback functions 0d6903e09914

## 6.6 [Low] Malicious user can inflate the price of vHype

**File(s)**: src/GenesisVaultManager.sol, src/VHYPE.sol

**Description**: User can deposit Hype tokens into GenesisVaultManager contract, against which he will receive VHYPE tokens. The user can burn his vHYPE tokens at his discretion by calling burn on the Token contract.

Burning the token will impact the totalSupply as the supply will reduce to the extent of tokens burnt. This will directly inflate the value of VHYPE tokens.

**Recommendation(s)**:Consider removing the burn function.

**Status**: Fixed

**Update from the client**: Not a bug per-se, since if a user intentionally burned their tokens, the exchange rate *should* get inflated. There isn't a way for an attacker to exploit this, since they would just end up losing their claim on the underlying HYPE in the net.

However, we discussed removing this burn function to avoid accidental burns and reduce the surface area of bugs. Removed in this PR 5d78a553ac7

## 6.7 [Low] _mint() can be called when the contract is paused

**File(s)**: src/VHYPE.sol

**Description**: The Base contract has a whenNotPaused modifier that is used to prevent certain actions when the contract is paused. The VHYPE contract inherits from Base and has a mint function. The mint function does not have whenNotPaused and hence VHYPE tokens can be minted even when VHYPE contract is paused.

```
1 function mint(address to, uint256 amount) public onlyManager {
2    _mint(to, amount);
3 }
```

**Recommendation(s)**: Consider adding the whenNotPaused modifier to the mint function. This would prevent new tokens from being minted when the contract is paused, ensuring consistent behavior across both public and internal minting methods.

**Status**: Fixed

**Update from the client**: Added whenNotPaused to mint() 5d4a5c7e88d

## 6.8 [Info] Event log can be cluttered with 0 deposits

**File(s)**: src/GenesisVaultManager.sol

**Description**: The `deposit` function can be called with 0 HYPE. As the deposit function does not check for msg.value to be greater than 0, the function will execute and generate a deposit event while consuming gas.

```
function deposit() public payable canDeposit whenNotPaused {
      uint256 requestedDepositAmount = msg.value;// if msg.value is 0
      //...


==>     emit Deposit(
          msg.sender, /* depositor */
          amountToMint, /* minted */
          amountToDeposit, /* deposited */
          requestedDepositAmount - amountToDeposit /* refunded */
      );
   }
```

**Recommendation(s)**: Consider adding a validation for `msg.value` to be greater than 0 or preferably a minimum deposit amount.

**Status**: Fixed

**Update from the client**: Fixed by minimum deposit limits 9007e61d162

## 6.9 [Info] Inconsistent comment for weiAmount conversion in CoreWriterLibrary

**File(s)**: src/lib/CoreWriterLibrary.sol

**Description**: The `CoreWriterLibrary` contract contains an inconsistent comment that states that weiAmount should be converted to 6 decimals before being passed to its functions. However, the rest of the protocol, including the `GenesisVaultManager` contract and the function-level Natspec comments, consistently uses 8 decimals for HYPE amounts sent to HyperCore. This discrepancy can lead to confusion.

The misleading comment is found in the contract-level NatSpec as below.

```
/// All weiAmounts must be converted to 6 decimals before calling these functions.
// ...
library CoreWriterLibrary {
   // ...
}
```

**Recommendation(s)**: Consider updating the `CoreWriterLibrary` contract's NatSpec comment to reflect the correct decimal conversion

**Status**: Fixed

**Update from the client**: Fixed 608e7fd69e4

## 6.10 [Info] emergencyStakingWithdraw may fail

**File(s)**: src/GenesisVaultManager.sol

**Description**: In the GenesisVaultManager contract, the funds moved to the HyperCore are delegated to a defaultVault.

The delegations to a particular validator have a lockup period of 1 day. Only after this lockup, delegations may be partially or fully undelegated at any time.

In the current implementation, the funds are delegated to the defaultVault.

If there arises a scenario where there is a need for emergencyStakingWithdraw, the earliest undelegate request can be submitted only after passing of 1 day from the last deposit.

```
function emergencyStakingWithdraw(uint256 amount, string calldata purpose) external onlyOwner {
      //...


==>   // Immediately undelegate HYPE
      // @audit, the undelagate can be called 1 day after
      // the last deposit.
       stakingVault.tokenDelegate(defaultValidator, amount.to8Decimals(), true);


     //...
   }
```

**Recommendation(s)**: Consider error handling in case emergencyStakingWithdraw does not work due to recent delegation to the validator.

**Status**: Fixed

**Update from the client**: Addressed in this f045057de3

## 6.11 [Info] whitelistDepositLimits should be greater than current defaultDepositLimit

**File(s)**: src/GenesisVaultManager.sol

**Description**: When setting the whitelistDepositLimit for an address, the limit should be greater than the current defaultDepositLimit.

```
function setDefaultDepositLimit(uint256 _defaultDepositLimit) public onlyOwner {
   defaultDepositLimit = _defaultDepositLimit;
}


function setWhitelistDepositLimit(address depositor, uint256 limit) public onlyOwner {
   whitelistDepositLimits[depositor] = limit;
}
```

**Recommendation(s)**: Consider adding a validation to check the limit to be greater than defaultDepositLimit.

**Status**: Fixed

**Update from the client**: Going to leave behavior as-is, but updated the comment to avoid confusion 19c95b5798

## 6.12 [Best Practices] Upgradeable contract lacks storage gaps

**File(s)**: src/Base.sol, *

**Description**: The protocol implements a Base contract using UUPS upgradeable proxy pattern. The common functionality is implemented in the Base contract, which is extended by other contracts in the protocol.

As the protocol follows the traditional storage for state, the storage layout is defined. In the future, if the Base contract needs to be extended to support more common features with state, it will not be possible. The absence of storage gaps removes the flexibility to insert variables in future upgrades without causing storage collisions.

**Recommendation(s)**: Consider adding storage gaps to provision future extension in functionality.

**Status**: Fixed

**Update from the client**: Fixed 0231aee978de

# 7 Issues in the new version

## 7.1 [Critical] Withdrawals can be claimed before finalisation

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

**Description**: The function `claimWithdraw()` performs following checks:

```
function claimWithdraw(uint256 withdrawId, address destination) public whenNotPaused {
        Withdraw storage withdraw = withdraws[withdrawId];
        require(msg.sender == withdraw.account, NotAuthorized());
        require(!withdraw.cancelled, WithdrawCancelled());
        require(!withdraw.claimed, WithdrawClaimed());

        Batch memory batch = batches[withdraw.batchIndex];
        require(block.timestamp > batch.finalizedAt + 7 days, WithdrawUnclaimable()); // TODO: Should we add a buffer?

        uint256 withdrawExchangeRate = batch.slashed ? batch.slashedExchangeRate : batch.snapshotExchangeRate;
        uint256 hypeAmount = _vHYPEtoHYPE(withdraw.vhypeAmount, withdrawExchangeRate);
        ...
}
```

The function, however, does not check if the `withdraw.batchIndex` is not `type(uint256).max`. Also, the requirement `block.timestamp > batch.finalizedAt + 7 days` does not take into consideration that `batch.finalizedAt` with value `0` would pass. When the request is processed but the batch is not finalised, the request would contain the current `batchIndex` and the batch would contain `snapshotExchangeRate`, but the `finalizedAt` would be `0`. This allows users to claim funds at the `snapshotExchangeRate` without the 7-day waiting period. In consequence, the calculations in the protocol would be incorrect, leading to loss of funds.

**Recommendation(s)**: Consider preventing claiming in waiting period by checking if the `withdraw.batchIndex` is not `type(uint256).max` and that `batch.finalizedAt` is not `0`.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/8def0bb173ed7df51196ad4ef3a61d5d9ce7ba83

## 7.2 [Medium] Lack of pause for `burnFrom()`

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

**Description**: The `burnFrom()` function in `VHYPE` does not contain the pause check.

**Recommendation(s)**: Consider introducing the pause check to `VHYPE.burnFrom()`.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/92b7aeb2a59a04db5d916bb22ec1f0b80ba8b610

## 7.3 [Medium] Lack of withdrawal request minimum amount can lead to DOS

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

**Description**: The function `queueWithdraw()` allows user to request withdrawal of any amount of `vHYP` greater than `0`. A malicious user can create requests with a small amount of `vHYPE` to temporarily DOS the batch finalisation. The finalisation can only be done if there are no withdrawal requests in the queue that could be fulfilled with available funds, which is checked in `_canFinalizeBatch()`. A malicious user could create a withdrawal request with a small amount before the finalisation, to add another request that can be fulfilled, effectively stopping the finalisation. Moreover, users could add many small-value requests to disincentivise calling `processBatch()` due to high gas cost.

**Recommendation(s)**: Consider defining minimum withdrawal request value.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/10e585a01ceee5acc995076163657adaadcca65e and https://github.com/ventuals/ventuals-contracts/commit/0847334cf30ca080af734a14d9318d2974b141e8

## 7.4 [Medium] Missing Validator address validation

**File(s)**: `StakingVault.sol`, `StakingVaultManager.sol`

**Description**: The `stake` and `tokenRedelegate` functions do not validate validator addresses. They will not fail with `validator == address(0)` or any invalid address, which could lead to silent errors since the functions will not revert but may fail silently during delegation operations.

**Recommendation(s)**: Add validation to ensure validator addresses are not zero addresses and consider implementing a whitelist of allowed validators.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/39e9bc8fec30658029fea4857ab96f5e41b2e870

## 7.5 [Medium] No limit on withdrawal request

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

**Description**: Users can request withdrawal of any amount. The requests are processed in a queue. Withdrawn amount is limited by the HyperCore staking restriction, at least `500k` HYPE tokens (initially) must be staked to maintain the perpetuals DEX deployment. This is checked in `StakingVaultManager` with `minimumStakeBalance` in the `processBatch()` function. However, there is a scenario where a single user requests a huge amount of tokens to withdraw. In such a case, there must be enough funds available to process the request of that user and all the subsequent requests. If that amount is really high, the withdrawals may be blocked for a substantial amount of time.

**Recommendation(s)**: Consider introducing maximum limit for withdrawal request. This would incentivise users to withdraw in smaller portions and allow continuous withdrawals.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/582641be06df9bc95d4bd9d5a7e53b2b6720248e. The fix is implemented such that we will automatically split a large withdraw request into smaller withdraws

## 7.6 [Medium] Problems with applySlash() after claiming withdrawals

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

**Description**: When the slashing happens, admins of the vHype protocol call the `applySlash()`. However, if this function is applied too late, users can claim the withdrawal funds from the affected batches. This may result in two problems:

- users can claim their withdrawals at the original (higher) exchange rate before slashing is applied, allowing them to circumvent slashing penalties;
- during the claiming the `totalHypeClaimed` is increased, but the `applySlash()` adjusts only the `totalHypeProcessed` according to the new `slashedExchangeRate`. This may result in `totalHypeClaimed > totalHypeProcessed` which would cause revery in `totalBalance()` since `reservedHypeForWithdraws = totalHypeProcessed - totalHypeClaimed` would result in underflow;

Note that the expected scenario is that the admins would react immediately to the slash event and call `applySlash()` only on the affected batches.

Recommendation(s): To ensure that no claiming happens between the actual slash event and calling `applySlash()`, consider introducing a two-step mechanism, which would first apply the slash or not, and only after allow claiming.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/ee2d169b8a12919d2babf5c04005b5d55bb5790b

## 7.7 [Medium] Truncation during HyperEVM to HyperCore transfer

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

**Description**: The truncation is not applied during the transfer of funds to the spot account in `finaliseBatch()`. This results in potential loss of `1e10 wei` during every deposit.

```
function finalizeBatch() external whenNotPaused whenBatchProcessingNotPaused {
      ...

        // Always transfer the full deposit amount to HyperCore spot
        if (depositsInBatch > 0) {
            stakingVault.transferHypeToCore(depositsInBatch);
        }
        ...
}
```

The `depositsInBatch` will be truncated during transfer to HyperCore, but the whole un-truncated amount will be deducted from the StakingVault.

**Recommendation(s)**: Consider applying restriction of precision less or equal to `1e8`. This can be applied by adding a check to enforce `amount % 1e10 == 0`.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/66961ade83d9be546527c46f594eba8a4a7571d9

## 7.8 [Low] Unclear error message when claiming unprocessed withdraw

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

**Description**: When `claimWithdraw` is called for a withdrawal that hasn't been processed yet (batchIndex is still `type(uint256).max`), the function doesn't fail early with a clear error message. Instead, it proceeds through all validation checks and only fails when reaching `spotSend()` because the balance will be 0, resulting in an unclear error message that doesn't indicate the withdrawal hasn't been processed yet.

**Recommendation(s)**: Add explicit validation to check that `withdraw.batchIndex != type(uint256).max` in `claimWithdraw` and provide a clear error message if the withdrawal hasn't been processed yet.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/8def0bb173ed7df51196ad4ef3a61d5d9ce7ba83

## 7.9 [Low] Validation logic should be moved to StakingVault

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

**Description**: The validation checks for Core user existence and spot balance are performed in `StakingVaultManager::claimWithdraw()` before calling `StakingVault::spotSend()`. These checks should be moved to the `StakingVault::spotSend()` function to ensure that if/when StakingVaultManager is upgraded, the essential validation checks are still in place since they are critical for StakingVault functionality.

**Recommendation(s)**: Move the Core user existence and spot balance validation checks to the `StakingVault::spotSend()` function to centralize validation logic.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/9b9cd8ab9e928a79e131ffb399d899dfd6928a69 and https://github.com/ventuals/ventuals-contracts/commit/3bc3b57526d9a195fe362ba292f37e6490f03850

## 7.10 [Info] Unclear timing logic in delegation reading

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

**Description**: The timing constraint logic uses `block.number > lastDelegationChangeBlockNumber[_validator] + 1` in the block number comparison, but it's unclear why this additional block is required. This creates inconsistency with other timing checks in the contract that don't use `+ 1` like in the deposit function, where we have `block.number > lastEvmToCoreTransferBlockNumber`.

**Recommendation(s)**: Clarify the reasoning for the `+ 1` requirement in the documentation or remove it for consistency.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/b59f8a984c12c76df5617418fbb455fe1e4c84c3

## 7.11 [Info] minimumStakeBalance changes can temporarily block batch finalization

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

**Description**: Increasing `minimumStakeBalance` after processing a batch but before finalization can temporarily prevent batch finalization if the new minimum makes `balance < minimumStakeBalance + hypeProcessed`. However, this can be resolved by lowering `minimumStakeBalance` again or using the `resetBatch` function.

**Recommendation(s)**: Consider adding a check in the `setMinimumStakeBalance` function to ensure that the new minimum is not greater than the current balance if a batch is already processing.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/65102fd2039a0620bbd284c021fe43e97e0ac9a1

## 7.12 [Best Practice] Increase timing checks

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

**Description**: There are two time checks in the manager: (1) a 7-day withdrawal queue and (2) a 1-day unstake lockup after stake. Those checks mirror the mechanisms on HyperCore. It is recommended to increase those checks slightly to ensure that the calls won't result in silent failures, which may cause loss of funds.

**Recommendation(s)**: Consider increasing time checks to ensure correct behaviour. Additionally, consider increasing the block separation time to be greater than 1.

**Status**: Fixed

**Update from the client**: Fixed https://github.com/ventuals/ventuals-contracts/commit/d7fea7e9116dbaf75443bacd26259340e24a3324

# 8 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 Ventuals documentation**

The Ventuals protocol documentation provides a comprehensive overview of the liquid staking token (LST) system built on Hyperliquid. The documentation effectively covers both the current implementation (Phase 1) and future planned features (Phase 2), providing appropriate context for the current audit. The team provided an overview of the main system components during the kick-off call with a detailed explanation of the intended functionalities and was responsive to questions during the review.

# 9 Test Suite Evaluation

## 9.1 Tests Output

```
[] Compiling...
[] Compiling 58 files with Solc 0.8.30
[] Solc 0.8.30 finished in 31.90s
Compiler run successful!

Ran 29 tests for test/RoleRegistry.t.sol:RoleRegistryTest
[PASS] test_AcceptOwnership_OnlyPendingOwner() (gas: 42750)
[PASS] test_GrantAndRevokeRole(bytes32,address) (runs: 256, : 59835, ~: 59835)
[PASS] test_GrantRole_NotOwner(bytes32,address) (runs: 256, : 22655, ~: 22655)
[PASS] test_GrantRole_OnlyOwner(bytes32) (runs: 256, : 70688, ~: 70688)
[PASS] test_GrantRole_SameUserMultipleRoles(bytes32,bytes32,address) (runs: 256, : 121036, ~: 121036)
[PASS] test_Initialize_CannotInitializeTwice() (gas: 18926)
[PASS] test_Initialize_DefaultPauseState(address) (runs: 256, : 12936, ~: 12936)
[PASS] test_Initialize_OwnerAndAdmin() (gas: 15593)
[PASS] test_PauseAndUnpause(address) (runs: 256, : 31916, ~: 31899)
[PASS] test_Pause_NotOwner(address,address) (runs: 256, : 27016, ~: 27016)
[PASS] test_Pause_OnlyOwner(address) (runs: 256, : 43894, ~: 43894)
[PASS] test_RenounceOwnership_NotOwner() (gas: 19184)
[PASS] test_RenounceOwnership_OnlyOwner() (gas: 21797)
[PASS] test_RevokeRole_NotOwner(bytes32,address) (runs: 256, : 77190, ~: 77190)
[PASS] test_RevokeRole_OneOfMultiple(bytes32,bytes32,address) (runs: 256, : 108114, ~: 108114)
[PASS] test_RevokeRole_OnlyOwner(bytes32) (runs: 256, : 58795, ~: 58795)
[PASS] test_RoleHolders_AfterRevoke() (gas: 147821)
[PASS] test_RoleHolders_EmptyRole() (gas: 13509)
[PASS] test_RoleHolders_MultipleUsers() (gas: 177570)
[PASS] test_RoleHolders_SingleUser() (gas: 71455)
[PASS] test_TransferOwnership_NewOwnerCanUpgrade() (gas: 1828300)
[PASS] test_TransferOwnership_NotOwner() (gas: 21571)
[PASS] test_TransferOwnership_OnlyOwner() (gas: 42349)
[PASS] test_Unpause_NotOwner(address,address) (runs: 256, : 45078, ~: 45078)
[PASS] test_Unpause_OnlyOwner(address) (runs: 256, : 35371, ~: 35354)
[PASS] test_UpgradeToAndCall_NotOwner() (gas: 918271)
[PASS] test_UpgradeToAndCall_NotOwnerWithData() (gas: 920198)
[PASS] test_UpgradeToAndCall_OnlyOwner() (gas: 917683)
[PASS] test_UpgradeToAndCall_WithData() (gas: 971199)
Suite result: ok. 29 passed; 0 failed; 0 skipped; finished in 315.83ms (1.14s CPU time)

Ran 57 tests for test/StakingVault.t.sol:StakingVaultTest
[PASS] test_AddApiWallet(address,string) (runs: 256, : 123364, ~: 130953)
[PASS] test_AddApiWallet_EmptyName(address) (runs: 256, : 94480, ~: 94480)
[PASS] test_AddApiWallet_NotOperator(address) (runs: 256, : 37066, ~: 37066)
[PASS] test_CannotFallback() (gas: 22847)
[PASS] test_CannotReceive() (gas: 23861)
[PASS] test_DelegatorSummary() (gas: 17986)
[PASS] test_Deposit_AsManager() (gas: 48625)
[PASS] test_Deposit_CanDepositInNextBlockAfterTransfer() (gas: 112120)
[PASS] test_Deposit_CanDepositWhenNoTransfersMade() (gas: 48591)
[PASS] test_Deposit_CannotDepositInSameBlockAsTransfer() (gas: 109752)
[PASS] test_Deposit_NotManager(address) (runs: 256, : 44186, ~: 44186)
[PASS] test_Deposit_WhenPaused() (gas: 72152)
[PASS] test_Deposit_ZeroAmount() (gas: 38046)
[PASS] test_FunctionsWhenPaused() (gas: 150240)
[PASS] test_SpotBalance() (gas: 19376)
[PASS] test_SpotSend_CoreUserDoesNotExist() (gas: 37861)
[PASS] test_SpotSend_InsufficientBalance() (gas: 40979)
[PASS] test_SpotSend_NotManager(address) (runs: 256, : 35949, ~: 35949)
[PASS] test_SpotSend_Success(address,uint64,uint64) (runs: 256, : 99975, ~: 99975)
[PASS] test_SpotSend_ZeroAmount() (gas: 35208)
[PASS] test_Stake_CanCallTwiceInSameBlock() (gas: 185908)
[PASS] test_Stake_NotManager(address) (runs: 256, : 38798, ~: 38798)
[PASS] test_Stake_Success(uint64) (runs: 256, : 144439, ~: 144439)
[PASS] test_Stake_ZeroAmount() (gas: 37341)
[PASS] test_TokenRedelegate_CannotCallAfterTokenDelegateInSameBlock() (gas: 142110)
[PASS] test_TokenRedelegate_CannotCallAfterTokenUndelegateInSameBlock() (gas: 145077)
[PASS] test_TokenRedelegate_CannotCallTwiceInSameBlock() (gas: 213371)
[PASS] test_TokenRedelegate_InsufficientBalance() (gas: 51620)
[PASS] test_TokenRedelegate_NotManager(address) (runs: 256, : 41006, ~: 41006)
[PASS] test_TokenRedelegate_SameValidator() (gas: 38748)
```

```
[PASS] test_TokenRedelegate_StakeLockedUntilFuture() (gas: 51713)
[PASS] test_TokenRedelegate_StakeUnlockedAtExactTimestamp() (gas: 207004)
[PASS] test_TokenRedelegate_Success(uint64) (runs: 256, : 211011, ~: 211011)
[PASS] test_TokenRedelegate_ValidatorNotFound() (gas: 46578)
[PASS] test_TokenRedelegate_WhenPaused() (gas: 68821)
[PASS] test_TokenRedelegate_ZeroAmount() (gas: 39504)
[PASS] test_TransferHypeToCore(uint256) (runs: 256, : 98876, ~: 98876)
[PASS] test_TransferHypeToCore_CanTransferInNextBlock() (gas: 111807)
[PASS] test_TransferHypeToCore_CannotTransferInSameBlock() (gas: 101328)
[PASS] test_TransferHypeToCore_InsufficientBalance() (gas: 39164)
[PASS] test_TransferHypeToCore_NotActivatedOnHyperCore() (gas: 39334)
[PASS] test_TransferHypeToCore_NotManager(address) (runs: 256, : 34761, ~: 34761)
[PASS] test_TransferHypeToCore_UpdatesLastTransferBlockNumber() (gas: 96924)
[PASS] test_TransferHypeToCore_WhenPaused() (gas: 63225)
[PASS] test_TransferHypeToCore_ZeroAmount() (gas: 40373)
[PASS] test_TransferOwnership_NewOwnerCanUpgrade() (gas: 3555616)
[PASS] test_Unstake_CannotCallAfterStakeInSameBlock() (gas: 138566)
[PASS] test_Unstake_CannotCallTwiceInSameBlock() (gas: 141010)
[PASS] test_Unstake_InsufficientBalance() (gas: 47045)
[PASS] test_Unstake_NotManager(address) (runs: 256, : 38589, ~: 38589)
[PASS] test_Unstake_StakeLockedUntilFuture() (gas: 48169)
[PASS] test_Unstake_StakeUnlockedAtExactTimestamp() (gas: 133217)
[PASS] test_Unstake_Success(uint64) (runs: 256, : 146365, ~: 146365)
[PASS] test_Unstake_ValidatorNotFound() (gas: 46144)
[PASS] test_Unstake_ZeroAmount() (gas: 38221)
[PASS] test_UpgradeToAndCall_NotOwner(address) (runs: 256, : 1786348, ~: 1786348)
[PASS] test_UpgradeToAndCall_OnlyOwner() (gas: 1787436)
Suite result: ok. 57 passed; 0 failed; 0 skipped; finished in 349.03ms (1.69s CPU time)

Ran 15 tests for test/VHYPE.t.sol:VHYPETest
[PASS] test_ApproveAndTransferFrom(uint256,uint256) (runs: 256, : 122828, ~: 122984)
[PASS] test_BurnFrom_InsufficientAllowance(address,address,uint256,uint256) (runs: 256, : 114264, ~: 114264)
[PASS] test_BurnFrom_WhenPaused() (gas: 142189)
[PASS] test_BurnFrom_WithApproval(address,address,uint256,uint256) (runs: 256, : 102217, ~: 102252)
[PASS] test_BurnFrom_WithoutApproval(address,address,uint256,uint256) (runs: 256, : 89895, ~: 89895)
[PASS] test_Burn_WhenPaused() (gas: 114654)
[PASS] test_CanBurnOwnTokens(address,uint256,uint256) (runs: 256, : 90738, ~: 90738)
[PASS] test_CanBurnWithApproval(address,address,uint256,uint256) (runs: 256, : 100307, ~: 100307)
[PASS] test_CannotBurnMoreThanBalance(address,uint256,uint256) (runs: 256, : 86265, ~: 86265)
[PASS] test_Mint_NotManager(address,uint256) (runs: 256, : 37204, ~: 37204)
[PASS] test_Mint_OnlyManager(address,uint256) (runs: 256, : 84264, ~: 84264)
[PASS] test_Transfer(uint256,uint256) (runs: 256, : 114093, ~: 114793)
[PASS] test_TransferOwnership_NewOwnerCanUpgrade() (gas: 2242961)
[PASS] test_UpgradeToAndCall_NotOwner(address) (runs: 256, : 1128974, ~: 1128974)
[PASS] test_UpgradeToAndCall_OnlyOwner() (gas: 1130629)
Suite result: ok. 15 passed; 0 failed; 0 skipped; finished in 816.39ms (1.32s CPU time)

Ran 133 tests for test/StakingVaultManager.t.sol:StakingVaultManagerTest
[PASS] test_ApplySlash_InvalidBatch() (gas: 34059)
[PASS] test_ApplySlash_MultipleBatches() (gas: 1072791)
[PASS] test_ApplySlash_NotOwner() (gas: 544193)
[PASS] test_ApplySlash_ReapplySlash() (gas: 678946)
[PASS] test_ApplySlash_ValidBatch() (gas: 678863)
[PASS] test_ApplySlash_ZeroSlashedExchangeRate() (gas: 633928)
[PASS] test_CancelWithdraw_AlreadyCancelled() (gas: 419135)
[PASS] test_CancelWithdraw_AlreadyProcessed() (gas: 561378)
[PASS] test_CancelWithdraw_InvalidWithdrawId() (gas: 34311)
[PASS] test_CancelWithdraw_NotAuthorized() (gas: 436863)
[PASS] test_CancelWithdraw_Success() (gas: 422257)
[PASS] test_CancelWithdraw_WhenContractPaused() (gas: 461329)
[PASS] test_CannotInitializeTwice() (gas: 28519)
[PASS] test_ClaimWithdraw_AlreadyClaimed() (gas: 756456)
[PASS] test_ClaimWithdraw_CoreUserDoesNotExist() (gas: 651804)
[PASS] test_ClaimWithdraw_ExactlySevenDays() (gas: 630954)
[PASS] test_ClaimWithdraw_InsufficientBalance() (gas: 652412)
[PASS] test_ClaimWithdraw_NotAuthorized() (gas: 629489)
[PASS] test_ClaimWithdraw_Success() (gas: 757961)
[PASS] test_ClaimWithdraw_TooEarly() (gas: 630382)
[PASS] test_ClaimWithdraw_WhenContractPaused() (gas: 657400)
[PASS] test_ClaimWithdraw_WithSlashedBatch() (gas: 805780)
[PASS] test_ClaimWithdraw_WithdrawCancelled() (gas: 420139)
[PASS] test_Deposit_BelowMinimumAmount() (gas: 117948)
[PASS] test_Deposit_EmptyVault() (gas: 179487)
[PASS] test_Deposit_ExactMinimumAmount() (gas: 202992)
[PASS] test_Deposit_ExchangeRateAboveOne() (gas: 216980)
```

```
[PASS] test_Deposit_ExchangeRateBelowOne() (gas: 219944)
[PASS] test_Deposit_RevertWhenContractPaused() (gas: 156435)
[PASS] test_Deposit_VaultWithExistingStakeBalance() (gas: 220180)
[PASS] test_Deposit_ZeroAmount() (gas: 109285)
[PASS] test_EmergencyStakingDeposit() (gas: 163613)
[PASS] test_EmergencyStakingDeposit_NotOwner() (gas: 32530)
[PASS] test_EmergencyStakingDeposit_ZeroAmount() (gas: 57328)
[PASS] test_EmergencyStakingWithdraw_InsufficientBalance() (gas: 69183)
[PASS] test_EmergencyStakingWithdraw_InsufficientDelegatedBalance() (gas: 70775)
[PASS] test_EmergencyStakingWithdraw_NotOwner() (gas: 32678)
[PASS] test_EmergencyStakingWithdraw_OnlyOwner() (gas: 173631)
[PASS] test_EmergencyStakingWithdraw_StakeLockedUntilFuture() (gas: 70930)
[PASS] test_EmergencyStakingWithdraw_StakeUnlockedAtExactTimestamp() (gas: 171918)
[PASS] test_EmergencyStakingWithdraw_ZeroAmount() (gas: 62813)
[PASS] test_ExchangeRate_AboveOne() (gas: 123947)
[PASS] test_ExchangeRate_BalanceLessThanSupply(uint256,uint256) (runs: 256, : 132919, ~: 132911)
[PASS] test_ExchangeRate_BalanceMoreThanSupply(uint256,uint256) (runs: 256, : 133907, ~: 133907)
[PASS] test_ExchangeRate_BelowOne() (gas: 124959)
[PASS] test_ExchangeRate_ZeroBalance() (gas: 126243)
[PASS] test_ExchangeRate_ZeroTotalSupply() (gas: 54616)
[PASS] test_FinalizeBatch_CanFinalizeAfterSlashIsApplied() (gas: 863419)
[PASS] test_FinalizeBatch_CanFinalizeWhenUnderMinimumStakeBalance() (gas: 599524)
[PASS] test_FinalizeBatch_CannotFinalizeAfterEmergencyStakingWithdrawInSameBlock() (gas: 773316)
[PASS] test_FinalizeBatch_CannotFinalizeAfterSlash() (gas: 635683)
[PASS] test_FinalizeBatch_CannotFinalizeAfterSwitchValidatorInSameBlock() (gas: 835931)
[PASS] test_FinalizeBatch_DepositsEqualWithdraws() (gas: 746713)
[PASS] test_FinalizeBatch_DepositsGreaterThanWithdraws() (gas: 856541)
[PASS] test_FinalizeBatch_DepositsLessThanWithdraws() (gas: 858807)
[PASS] test_FinalizeBatch_DustHypeAmountNotTransferred() (gas: 747065)
[PASS] test_FinalizeBatch_ZeroDeposits() (gas: 735683)
[PASS] test_FinalizeBatch_ZeroDepositsZeroWithdraws() (gas: 303621)
[PASS] test_FinalizeBatch_ZeroWithdraws() (gas: 462504)
[PASS] test_FinalizeResetBatch_FailsWithFinalizedBatch() (gas: 741537)
[PASS] test_FinalizeResetBatch_FailsWithNoBatch() (gas: 35672)
[PASS] test_FinalizeResetBatch_FailsWithPartialReset() (gas: 769626)
[PASS] test_FinalizeResetBatch_NotOwner() (gas: 524767)
[PASS] test_FinalizeResetBatch_Success() (gas: 502626)
[PASS] test_GetWithdrawAmount_ProcessedWithdraw() (gas: 539082)
[PASS] test_GetWithdrawAmount_SlashedBatch() (gas: 581987)
[PASS] test_GetWithdrawAmount_UnprocessedWithdraw() (gas: 436132)
[PASS] test_GetWithdrawClaimableAt_Finalized() (gas: 628649)
[PASS] test_GetWithdrawClaimableAt_NotFinalized() (gas: 534915)
[PASS] test_GetWithdrawClaimableAt_UnprocessedWithdraw() (gas: 420918)
[PASS] test_GetWithdrawClaimableAt_WithCustomClaimWindowBuffer() (gas: 633263)
[PASS] test_HYPETovHYPE_ExchangeRateAboveOne() (gas: 131120)
[PASS] test_HYPETovHYPE_ExchangeRateBelowOne() (gas: 130262)
[PASS] test_HYPETovHYPE_OneExchangeRate_ZeroSupply() (gas: 62140)
[PASS] test_HYPETovHYPE_ZeroAmount() (gas: 129467)
[PASS] test_HYPETovHYPE_ZeroExchangeRate_ZeroBalance() (gas: 128095)
[PASS] test_HYPETovHYPE_vHYPEtoHYPE_Roundtrip(uint256,uint256,uint256) (runs: 256, : 161810, ~: 161995)
[PASS] test_Initialize() (gas: 38298)
[PASS] test_ProcessBatch_CancelledWithdrawsSkipped() (gas: 801041)
[PASS] test_ProcessBatch_EmptyQueue() (gas: 227658)
[PASS] test_ProcessBatch_FirstBatch() (gas: 551313)
[PASS] test_ProcessBatch_InsufficientCapacity() (gas: 521819)
[PASS] test_ProcessBatch_MultipleWithdraws() (gas: 995460)
[PASS] test_ProcessBatch_PartialProcessing() (gas: 796206)
[PASS] test_ProcessBatch_UnderMinimumStakeBalance() (gas: 529696)
[PASS] test_ProcessBatch_WhenBatchProcessingPaused() (gas: 450601)
[PASS] test_ProcessBatch_WhenContractPaused() (gas: 449242)
[PASS] test_ProcessBatch_WithOneDayLockedStake() (gas: 972548)
[PASS] test_ProcessBatch_WithOneDayLockedStakeAndExtraTime() (gas: 870893)
[PASS] test_QueueWithdraw_BelowMinimumAmount() (gas: 168315)
[PASS] test_QueueWithdraw_InsufficientAllowance() (gas: 175863)
[PASS] test_QueueWithdraw_InsufficientBalance() (gas: 172052)
[PASS] test_QueueWithdraw_MultipleWithdraws() (gas: 660016)
[PASS] test_QueueWithdraw_Success() (gas: 441038)
[PASS] test_QueueWithdraw_WhenContractPaused() (gas: 197233)
[PASS] test_QueueWithdraw_ZeroAmount() (gas: 31369)
[PASS] test_ResetBatch_CannotResetFinalizedBatch() (gas: 624769)
[PASS] test_ResetBatch_NotOwner() (gas: 539889)
[PASS] test_ResetBatch_NothingToReset() (gas: 34768)
[PASS] test_ResetBatch_Success() (gas: 1042351)
[PASS] test_ResetBatch_WithCancelledWithdrawals() (gas: 1042110)
[PASS] test_SetBatchProcessingPaused_NotOwner() (gas: 29866)
```

```
[PASS] test_SetBatchProcessingPaused_OnlyOwner() (gas: 58622)
[PASS] test_SetClaimWindowBuffer_NotOwner() (gas: 29611)
[PASS] test_SetClaimWindowBuffer_OnlyOwner() (gas: 35979)
[PASS] test_SetClaimWithdrawBuffer_UpdatesClaimWithdrawValidation() (gas: 631904)
[PASS] test_SetMinimumDepositAmount_NotOwner() (gas: 31481)
[PASS] test_SetMinimumDepositAmount_OnlyOwner() (gas: 38071)
[PASS] test_SetMinimumDepositAmount_UpdatesDepositValidation() (gas: 224951)
[PASS] test_SetMinimumStakeBalance_NotOwner() (gas: 33159)
[PASS] test_SetMinimumStakeBalance_OnlyOwner() (gas: 41048)
[PASS] test_SetMinimumStakeBalance_TooLowDuringBatchProcessing() (gas: 564013)
[PASS] test_SetMinimumWithdrawAmount_NotOwner() (gas: 30139)
[PASS] test_SetMinimumWithdrawAmount_OnlyOwner() (gas: 36666)
[PASS] test_SetMinimumWithdrawAmount_UpdatesWithdrawValidation() (gas: 485348)
[PASS] test_SpotAccountBalance() (gas: 32622)
[PASS] test_StakingAccountBalance() (gas: 31946)
[PASS] test_SwitchValidator_NotOwner() (gas: 40787)
[PASS] test_SwitchValidator_OnlyOwner() (gas: 236045)
[PASS] test_SwitchValidator_SameValidator() (gas: 68040)
[PASS] test_SwitchValidator_StakeLockedUntilFuture() (gas: 76649)
[PASS] test_SwitchValidator_StakeUnlockedAtExactTimestamp() (gas: 233256)
[PASS] test_TotalBalance() (gas: 43782)
[PASS] test_TotalBalance_ReservedHypeForWithdrawsGreaterThanAccountBalance() (gas: 652737)
[PASS] test_TotalBalance_WithReservedHypeForWithdraws() (gas: 700127)
[PASS] test_TransferOwnership_NewOwnerCanUpgrade() (gas: 5604808)
[PASS] test_UpgradeToAndCall_NotOwner() (gas: 2799309)
[PASS] test_UpgradeToAndCall_OnlyOwner() (gas: 2811124)
[PASS] test_vHYPEtoHYPE_ExchangeRateAboveOne() (gas: 130225)
[PASS] test_vHYPEtoHYPE_ExchangeRateBelowOne() (gas: 130049)
[PASS] test_vHYPEtoHYPE_ZeroAmount() (gas: 129474)
[PASS] test_vHYPEtoHYPE_ZeroExchangeRate_ZeroBalance() (gas: 129021)
[PASS] test_vHYPEtoHYPE_ZeroExchangeRate_ZeroSupply() (gas: 60057)
Suite result: ok. 133 passed; 0 failed; 0 skipped; finished in 816.40ms (1.20s CPU time)

Ran 4 test suites in 832.36ms (2.30s CPU time): 234 tests passed, 0 failed, 0 skipped (234 total tests)
```

## 9.2 Automated Tools

### 9.2.1 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.

# 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 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 conducts cutting-edge internal research and collaborates closely with external partners on cryptographic protocols, consensus design, succinct arguments and folding schemes, elliptic curve-based STARK protocols, post-quantum security and zero-knowledge proofs (ZKPs). Our research has led to influential contributions, including Zinc (Crypto '25), Mova, FLI (Asiacrypt '24), and foundational results in Fiat-Shamir security and STARK proof batching. Complementing this theoretical work, our engineering expertise is demonstrated through implementations such as the Latticefold aggregation scheme, the Labrador proof system, zkvm-benchmarks, and Plonk Verifier in Cairo. This combined strength in theory and engineering enables us to deliver cutting-edge cryptographic solutions to 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.

### 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.