Security Review Report
NM-0724 Obol

NETHERMIND SECURITY

(Dec 1, 2025)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for Obol Network's Ethereum validator manager smart contracts. The audit focused on changes introduced on PR #184 which sought to make the `Obol Validator Manager` compatible with Liquid-collective's `TVS` contract. The `ObolValidatorManager` contract is responsible for managing Ethereum validators and distributing payments between principal and reward recipients.

The audit comprises 417 lines of code written in Solidity language. It focused on the **ObolValidatorManagerFactory**, **ObolValidatorManager** smart contracts and the **IObolValidatorManager** interface contract.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases. **Along this document, we report** two points of attention, one classified as `High` and one as `Informational`. The issues are summarized in Fig. 1.

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

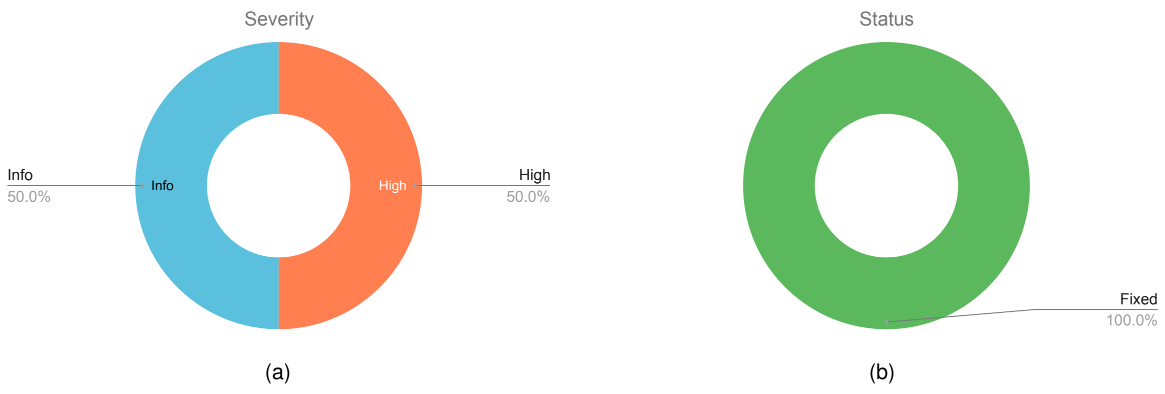

Fig. 1: Distribution of issues: Critical (0), **High** (1), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (1), **Best Practices** (0).
Distribution of status: Fixed (0), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (2)

Summary of the Audit

Audit Type	Security Review
Initial Report	November 27, 2025
Response from Client	Regular responses during audit engagement
Final Report	December 1, 2025
Repository	PR #184
Initial Commit	6ffa33e
Final Commit	fb8e178
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/ovm/ObolValidatorManagerFactory.sol	50	45	90%	17	112
2	src/ovm/ObolValidatorManager.sol	294	124	42%	96	510
3	src/interfaces/IObolValidatorManager.sol	73	158	216%	59	290
	Total	**417**	**327**	**78.4%**	**172**	**912**

3 Summary of Issues

	Finding	Severity	Update
1	Inconsistent `fundsPendingWithdrawal` Accounting in `sweep(...)` Function	High	Fixed
2	`ObolValidatorManager` is not fully compatible with the `ITVS` interface	Info	Fixed

4 System Overview

Obol built a set of smart contracts designed to manage Ethereum validators and distribute staking rewards and withdrawals between principal and reward recipients. The protocol consists of two main contracts:

- **ObolValidatorManagerFactory**: This contract acts as a factory, enabling the deployment of multiple `ObolValidatorManager` instances through the `createObolValidatorManager` function.
- **ObolValidatorManager**: This contract serves as the withdrawal address for validators. It is responsible for implementing the logic that handles the distribution of staking rewards and the management of validator withdrawals.

4.1 Pull Request (PR) #184

`PR #184` sought to make the `Obol Validator Manager` contract compatible with `Liquid-collectiveś TVS` contract. As a result, the `Obol Validator Manager` contract was refactored.

Majorly, the refactored code updated:

4.1.1 Validator Consolidation

Validator consolidation now happens via the `consolidate(...)` function using `EIP7251`. The function takes a `struct` of the consolidation request as input with public keys of `validators` to consolidate.

```
struct ConsolidationRequest {
    bytes[] srcPubKeys;
    bytes targetPubKey;
  }
```

4.1.2 Sweeping tokens

The `sweep(...)` function allows a user to sweep a specific or all pending withdrawal funds to either the contract set `beneficiary` or a user chosen recipient.

```
function sweep(address beneficiary, uint256 amount) external;
```

4.1.3 Ownership Transfer

The `Obol Validator Manager` contract's ownership can now be transferred to a new owner.

```
function transfer(address newBeneficiary, address newOwner) external;
```

Additionally, smart contract `events` and `errors` were updated to reflect the refactored code.

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] Inconsistent `fundsPendingWithdrawal` Accounting in `sweep(...)` Function

File(s): `ObolValidatorManager.sol`

Description: The `sweep(...)` function decrements `pullBalances[principalRecipient]` when funds are swept, but fails to update `fundsPendingWithdrawal` accordingly. This breaks the critical invariant that `fundsPendingWithdrawal` should equal the sum of all `pullBalances` across all accounts, as indicated by the comment in `withdrawPullBalance()` creating an accounting discrepancy. As a result, `fundsPendingWithdrawal` becomes inflated, leading to incorrect fund distribution.

```
// sweep() - MISSING fundsPendingWithdrawal update
function sweep(address beneficiary, uint256 amount) external nonReentrant {
    // ...
    pullBalances[principalRecipient] -= sweepAmount;  // Line 172
    // Missing: fundsPendingWithdrawal -= sweepAmount;
    recipient.safeTransferETH(sweepAmount);
}

// withdrawPullBalance() - CORRECT implementation
function withdrawPullBalance(address account) external {
    uint256 amount = pullBalances[account];
    if (amount == 0) return;

    unchecked {
        fundsPendingWithdrawal -= uint128(amount);  // Line 281 - CORRECT
    }
    pullBalances[account] = 0;
    account.safeTransferETH(amount);
}

// _distributeFunds() - Uses fundsPendingWithdrawal for calculation
function _distributeFunds(uint256 pullOrPush) internal {
    uint256 _memoryFundsPendingWithdrawal = uint256(fundsPendingWithdrawal);
    uint256 _fundsToBeDistributed = currentbalance - _memoryFundsPendingWithdrawal;  // Line 443
    // ...
}
```

Recommendation(s): Consider update `sweep(...)` to decrement `fundsPendingWithdrawal` when funds are swept, mirroring the behavior in `withdrawPullBalance(...)`

Status: Fixed

Update from the client: Fixed here

6.2 [Info] `ObolValidatorManager` is not fully compatible with the `ITVS` interface

File(s): `ObolValidatorManager.sol`

Description: The `ObolValidatorManager` contract is intended to conform to the `ITVS` interface. However, it does not implement the `version` function defined in the interface, resulting in incomplete compliance.

Recommendation(s): Consider implementing the missing version function to ensure full compatibility with the `ITVS` interface.

Status: Fixed

Update from the client: Fixed here

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 Obol's documentation

The **Obol** team provided a walkthrough of the PR in scope during the kickoff call. The team addressed the questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the new changes related to the PR.

8 Test Suite Evaluation

8.1 Tests Output

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

Ran 44 tests for src/test/ovm/ObolValidatorManager.t.sol:ObolValidatorManagerTest
[PASS] testBatchConsolidation() (gas: 1363028)
[PASS] testBatchWithdrawal() (gas: 1230879)
[PASS] testCan_distributeDirectDepositsAsReward() (gas: 108481)
[PASS] testCan_distributeMultipleDepositsToBeneficiaryRecipient() (gas: 93425)
[PASS] testCan_distributeMultipleDepositsTorewardsRecipient() (gas: 84595)
[PASS] testCan_distributePushAndPull() (gas: 199014)
[PASS] testCan_distributeToBothRecipients() (gas: 140780)
[PASS] testCan_distributeToNoRecipients() (gas: 24837)
[PASS] testCan_distributeToPullFlow() (gas: 164420)
[PASS] testCan_distributeToSecondRecipient() (gas: 81670)
[PASS] testCan_emitOnDistributeToNoRecipients() (gas: 22330)
[PASS] testCan_recoverNonOVMFundsToRecipient() (gas: 237936)
[PASS] testCannotDeposit() (gas: 62442)
[PASS] testCannot_consolidate() (gas: 1040514052)
[PASS] testCannot_distributeTooMuch() (gas: 101663)
[PASS] testCannot_recoverFundsToNonRecipient() (gas: 42012)
[PASS] testCannot_reenterOVM() (gas: 4339221)
[PASS] testCannot_setBeneficiaryRecipient() (gas: 52791)
[PASS] testCannot_setRewardRecipient() (gas: 52676)
[PASS] testCannot_sweepAfterRenounceOwnership() (gas: 87347)
[PASS] testCannot_sweepMoreThanBalance() (gas: 86603)
[PASS] testCannot_sweepMoreThanPullBalance() (gas: 104423)
[PASS] testCannot_sweepToCustomAddressAsNonOwner() (gas: 88438)
[PASS] testCannot_withdraw() (gas: 1040450038)
[PASS] testDefaultParameters() (gas: 22769)
[PASS] testDeposit() (gas: 50663)
[PASS] testFuzzCan_distributeDepositsToRecipients(uint64,uint8,uint256,uint256) (runs: 100, : 3071302, ~: 3038134)
[PASS] testFuzzCan_distributePullDepositsToRecipients(uint64,uint8,uint256,uint256) (runs: 100, : 3149385, ~: 3105311)
[PASS] testOwnerInitialization() (gas: 11302)
[PASS] testReceiveERC20() (gas: 42554)
[PASS] testReceiveETH() (gas: 15166)
[PASS] testReceiveTransfer() (gas: 15226)
[PASS] testSetAmountOfPrincipalStake() (gas: 32550)
[PASS] testSetBeneficiaryRecipient() (gas: 27838)
[PASS] testSetRewardRecipient() (gas: 27864)
[PASS] testSingleConsolidation() (gas: 235917)
[PASS] testSingleWithdrawal() (gas: 204241)
[PASS] testSweep_allFunds() (gas: 106544)
[PASS] testSweep_asNonOwnerToDefaultBeneficiary() (gas: 108044)
[PASS] testSweep_partialAmount() (gas: 126224)
[PASS] testSweep_toDefaultBeneficiary() (gas: 106323)
[PASS] testSweep_toSpecificAddress() (gas: 110414)
[PASS] test_Transfer_HappyPath() (gas: 44952)
[PASS] test_WithdrawZeroBalance() (gas: 14120)
Suite result: ok. 44 passed; 0 failed; 0 skipped; finished in 42.88ms (88.22ms CPU time)

Ran 8 tests for src/test/ovm/ObolValidatorManagerFactory.t.sol:ObolValidatorManagerFactoryTest
[PASS] testCan_createOVM() (gas: 2923571)
[PASS] testCan_emitOnCreate() (gas: 5824551)
[PASS] testCannot_createWithInvalidOwner() (gas: 14666)
[PASS] testCannot_createWithInvalidRecipients() (gas: 23354)
[PASS] testCannot_createWithInvalidThreshold() (gas: 22247)
[PASS] testFuzzCan_createOVM(uint64) (runs: 100, : 2918372, ~: 2918372)
[PASS] testFuzzCannot_CreateWithLargeThreshold(address,uint64) (runs: 100, : 14059, ~: 14059)
[PASS] testFuzzCannot_CreateWithZeroThreshold(address) (runs: 100, : 18599, ~: 18599)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 119.77ms (129.14ms CPU time)

Ran 2 test suites in 123.48ms (162.65ms CPU time): 52 tests passed, 0 failed, 0 skipped (52 total tests)
```

8.2 Automated Tools

8.2.1 AuditAgent

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 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