Security Review Report
NM-0482 Sophon

NETHERMIND
SECURITY

(May 30, 2025)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for Sophon smart contracts. Sophon community participates in the protocol by running nodes for the network, for which they earn rewards. Instead of giving vSophon tokens immediately on accrual, the rewards are wrapped in WSophon tokens for vesting. The rewards are released to the participant when they are fully vested.

There is a weekly process owned by the protocol team, that adds the vesting schedule with accrued earnings for all participants. Participants can claim their earnings after the expiration of the vesting period

The audit comprises 327 lines of solidity code. **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** three points of attention, where all three are classified as `Informational` or **Best Practices**. The issues are summarized in Fig. 1.

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

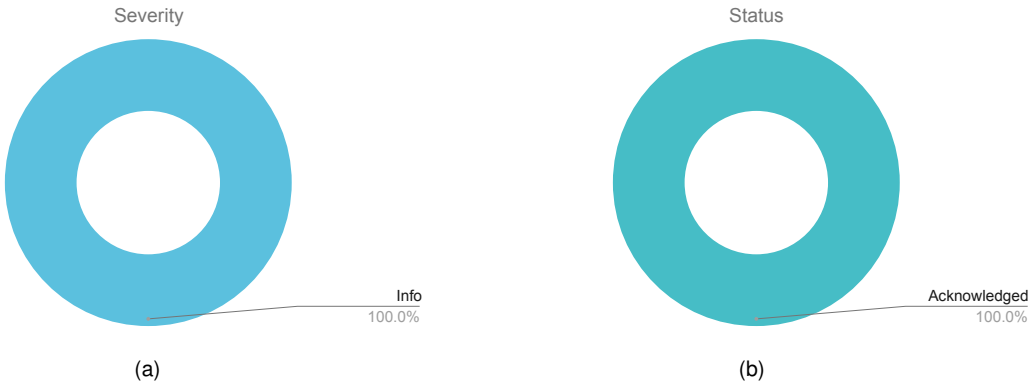


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (3), **Best Practices** (0). **Distribution of status: Fixed** (0), **Acknowledged** (3), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	April 7, 2025
Final Report	May 30, 2025
Repository	sophon-smart-contracts
Commit	1480b64b3b41a73c6c9427885ddc7220152471e7
Final Commit	1480b64b3b41a73c6c9427885ddc7220152471e7
Documentation	Low
Documentation Assessment	Low
Test Suite Assessment	Low

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	VSophTokenState.sol	13	9	69.2%	5	27
2	VSophToken.sol	314	172	54.8%	88	574
	Total	**327**	**181**	**55.4%**	**93**	**601**

3 Summary of Issues

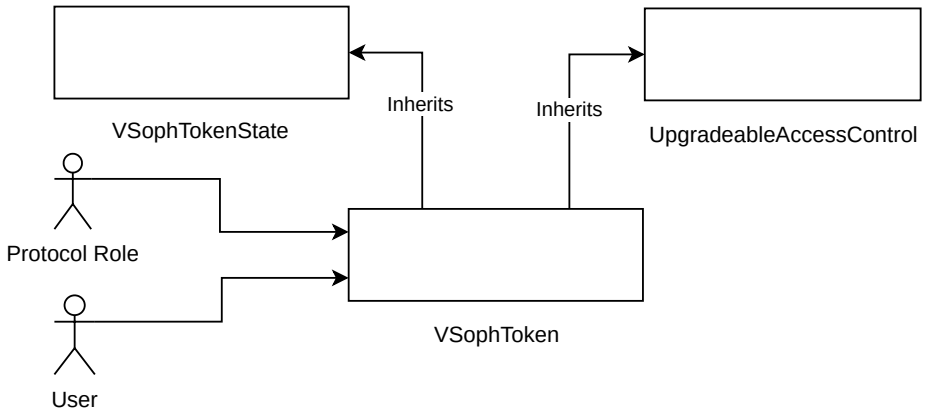
	Finding	Severity	Update
1	Releasing of vesting schedules in blocks could revert	Info	Acknowledged
2	`_transferAllVestingSchedules` transfers even fully vested schedules	Info	Acknowledged
3	`adminAddress` should be validated for non zero address	Info	Acknowledged

4 System Overview

The **Sophon** smart contracts in the scope of this audit are designed to manage the releasing of accrued reward tokens for node validators. The smart contracts primarily focus on adding the accrued rewards for the network participants to vesting schedules. Participants will subsequently claim the tokens once they are completely vested.

The schedules for accrued rewards are added to the vesting schedules by a weekly process, which is owned by the protocol. The users will be able to claim the rewards when they are fully vested.

The smart contracts also have a provision to transfer the vested schedules from one account to another. This functionality is added to handle any unexpected circumstances.



VSophToken contract is the main user interfacing contract. It contains all user facing functions for interacting with the Vesting Schedules. It handles the adding of new schedules, releasing fully vested tokens, and transfer of vesting schedules from one account to another.

4.1 Adding Vesting Schedule

The accrued rewards for the participants are added to the vesting schedule by a weekly process. The process will call one of the below functions to add Vesting schedules for the participants. Adding of vesting schedules is managed by the protocol via the schedule manager role.

- **addVestingSchedule**: Adds a vesting schedule for a single beneficiary, transferring wSoph from the vault and unwrapping it.
- **addMultipleVestingSchedules**: Works like **addVestingSchedule**, but adds multiple vesting schedules at once.

4.2 Releasing Vesting Schedule

While creating new vesting schedules is managed by the schedule manager role, claiming the fully vested tokens is the responsibility of the participant.

- **releaseSpecificSchedules**: Using this function, the user can claim fully vested tokens from specific schedules that are provided in an array.
- **releaseAllSchedules**: Releases vested tokens for all schedules.
- **releaseSchedulesInRange**: Releases vested tokens from a range of schedules.

4.3 Transfer of vesting schedules

VSophToken contract contains functions to transfer vesting schedules from one account to another. These functions are to handle un expected circumstances.

- **transferAllVestingSchedules**: Transfers all vesting schedules from a source account to a target account.
- **transferVestingSchedule**: Transfers a single schedule by index from a source account to a target account.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Info] Releasing of vesting schedules in blocks could revert

File(s): VSophToken.sol

Description: VSophToken contract supports releasing of vesting schedules in blocks through releaseSchedulesInRange and releaseAllSchedules functions.

```
function releaseSchedulesInRange(uint256 startIndex, uint256 endIndex) external {
    VestingSchedule[] storage schedules = vestingSchedules[msg.sender];
    _releaseSchedulesInRange(schedules, startIndex, endIndex);
}

function releaseAllSchedules() external {
    VestingSchedule[] storage schedules = vestingSchedules[msg.sender];
    _releaseSchedulesInRange(schedules, 0, schedules.length);
}
```

These two functions accept a block of vesting schedules to be released in a single call. These two functions internally calls _releaseSchedulesInRange function based on start and end indexes qualified for the schedules to be processed for release. As the start dates for all the vesting schedules are not organised in chronological order, it is possible for one of the schedules to have a start date beyond the current block time stamp.

```
function _releaseSchedulesInRange(VestingSchedule[] storage schedules, uint256 startIndex, uint256 endIndex)
    internal
{
    if (schedules.length == 0) revert NoVestingSchedule();
    if (startIndex >= endIndex || endIndex > schedules.length) revert InvalidRange();

    uint256 totalAmountToRelease = 0;

    uint256 vestingStartDate_ = vestingStartDate;

    for (uint256 i = startIndex; i < endIndex; i++) {
==>     uint256 released = _processSchedule(schedules[i], vestingStartDate_);
        totalAmountToRelease += released;
    }

    if (totalAmountToRelease == 0) revert NoTokensToRelease();
}
```

So, while processing each schedule by calling _processSchedule, if the start time of the schedule happens to be a future time, meaning the vesting schedule did not start yet. In such case, the transaction will revert blocking all the other vesting schedules that were eligible to be released.

```
function _processSchedule(VestingSchedule storage schedule, uint256 vestingStartDate_)
    internal
    returns (uint256 releasedAmount)
{
    // this is critical part. set startDate if schedule.startDate was zero
    if (vestingStartDate_ != 0 && block.timestamp >= vestingStartDate_ && schedule.startDate == 0) {
        schedule.startDate = vestingStartDate_;
    }

==> if (!_hasVestingStarted(schedule)) revert VestingHasNotStartedYet();

    //...
}
```

Recommendation(s): return 0 for vesting schedule where vesting has not yet started. This will allow release of tokens for other vesting schedules.

Status: Acknowledged

Update from the client:

6.2 [Info] `_transferAllVestingSchedules` transfers even fully vested schedules

File(s): VSophToken.sol

Description: The `_transferAllVestingSchedules` function is transferring all vesting schedules from one account to another, including those that are fully vested. The fully vested schedule does not have any pending amounts for future releases.

```
function _transferAllVestingSchedules(address from, address to) internal {
    if (to == address(0) || from == address(0) || from == to) revert InvalidRecipientAddress();
    if (vestingSchedules[from].length == 0) revert NoVestingSchedule();

    VestingSchedule[] storage fromSchedules = vestingSchedules[from];

    // Move vesting schedules
    for (uint256 i = 0; i < fromSchedules.length; i++) {
        vestingSchedules[to].push(fromSchedules[i]);
    }

    // --SNIP
```

Transferring fully vested schedules is redundant and unnecessary, as the tokens associated with them are no longer releasable. This leads to unnecessary growth of the `vestingSchedules` array for the recipient.

Recommendation(s): Modify the function to transfer only schedules that are not fully vested. This can be achieved by checking whether the schedule has any remaining releasable amount before transferring.

Status: Acknowledged

Update from the client:

6.3 [Info] `adminAddress` should be validated for non zero address

File(s): VSophToken.sol

Description: In the `initialize` function, the `adminAddress` should be validated to be non zero as this account is entitled with many roles.

Recommendation(s): Validate `adminAddress` to be a non zero address.

Status: Acknowledged

Update from the client:

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 the Sophon documentation

The Sophon team was actively present in regular calls, effectively addressing concerns and questions raised by the Nethermind Security team. Additionally, the code includes NatSpec documentation for different functions and their parameters. However, the project documentation could be improved by providing a more comprehensive written overview of the system and an explanation of the different design choices.

8 Complementary Checks

8.1 Compilation Output

None

8.2 Tests Output

None

8.3 Automated Tools

8.3.1 AuditAgent

All the relevant issues raised by the AuditAgent have been incorporated into this report. The AuditAgent is an AI-powered smart contract auditing tool that analyses code, detects vulnerabilities, and provides actionable fixes. It accelerates the security analysis process, complementing human expertise with advanced AI models to deliver efficient and comprehensive smart contract audits. Available at https://app.auditagent.nethermind.io.

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.