OMNISCIA

May 28, 2025

SMART CONTRACT AUDIT REPORT

Sophon Network
Staking Merkle Contracts


omniscia.io


info@omniscia.io


Online report: sophon-network-staking-merkle-contracts

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.


omniscia.io


info@omniscia.io

Online report: sophon-network-staking-merkle-contracts

Staking Merkle Contracts Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
7084f1cc99	May 8th 2025	4cc0ef64ad
e04476671d	May 20th 2025	dd36d22098
e04476671d	May 20th 2025	8f49590f58
564f8b9293	May 28th 2025	43162b95d3

Audit Overview

We were tasked with performing an audit of the Sophon Network codebase and in particular their Staking & Merkle Contracts.

The system implements a staking system for validators that utilizes a complex per-validator fee system imposed on any rewards by maintaining a validator profit numerator.

Additionally, a Merkle-proof based claim module is present in the codebase that permits users to claim rewards as well as stake them in the same call if they wish so.

Over the course of the audit, we identified a severe flaw when switching validators within the staking system that could have resulted in total fund loss of any validator's stake.

Additionally, we observed several dangerous arithmetic approaches in the current design of the staking system that are prone to errors and can result in unforeseen consequences, one of which we materialized in a Denial-of-Service attack.

We advise the Sophon Network team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Sophon Network team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Sophon Network and have identified that certain exhibits have not been adequately dealt with. We advise the Sophon Network team to revisit the following exhibits: `SSG-02M`, `SSG-03M`, `SSG-04M`

Additionally, the description of the following exhibit has been revised and should be revisited: `MCR-02C`

Post-Audit Conclusion (564f8b9293)

The Sophon Network team evaluated our follow-up recommendations and proceeded with alleviating all exhibits except for `SSG-04M` for which further justification as to its acknowledgement was provided.

We consider all outputs of the audit report properly consumed by the Sophon Network team with no outstanding remediative actions remaining.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	9	9	0	0
Minor	2	2	0	0
Medium	5	4	0	1
Major	2	2	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **16 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

- **Scope**
- **Compilation**
- **Static Analysis**
- **Manual Review**
- **Code Style**

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/sophon-org/support-contracts**
- Commit: 7084f1cc994d1e0878274dc650b4edeb7901a945
- Language: Solidity
- Network: Sophon
- Revisions: **7084f1cc99**, **e04476671d**, **564f8b9293**

Contracts Assessed

File	Total Finding(s)
contracts/tokens/staking/MerkleClaimer.sol (MCR)	6
contracts/tokens/staking/MerkleClaimerProxy.sol (MCP)	0
contracts/tokens/staking/SophonStaking.sol (SSG)	10
contracts/tokens/staking/SophonStakingProxy.sol (SSP)	0
contracts/tokens/staking/SophonStakingState.sol (SSS)	2
contracts/tokens/staking/SophonStakingSignals.sol (SSL)	0

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.28` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used, however, they are solely contained in external dependencies and can thus be safely ignored.

The `pragma` statements have been locked to `0.8.28` (`=0.8.28`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **39 potential issues** within the codebase of which **36 were ruled out to be false positives** or negligible findings.

The remaining **3 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
MCR-01S	Informational	Yes	Inexistent Sanitization of Input Address
SSG-01S	Informational	Yes	Illegible Numeric Value Representations

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Sophon Network's staking and Merkle-proof claim system.

As the project at hand implements a novel staking implementation, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed multiple significant vulnerabilities** within the system which could have had **moderate-to-severe ramifications** to its overall operation; for more information, kindly consult the audit report's summary as well as the dedicated medium and high severity exhibits within.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be.

A total of **16 findings** were identified over the course of the manual review of which **9 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
MCR-01M	Minor	Yes	Fixed Domain Separator Calculation
MCR-02M	Medium	Yes	Improper Nonce Increments
MCR-03M	Medium	Yes	Inexistent Restriction of Batch Operations
SSG-01M	Minor	Yes	Discrepancy of Contract Initialization
SSG-02M	Medium	Yes	Incorrect Assumption of Staked Funds

SSG-03M	Medium	Yes	Inexistent Prevention of Validator Profit Numerator Underflow
SSG-04M	Medium	Acknowledged	Significantly Error-Prone Mathematical Calculations
SSG-05M	Major	Yes	Incorrect Validator Registration Mechanism
SSG-06M	Major	Yes	Staked Fund Duplication Attack

Code Style

During the manual portion of the audit, we identified **7 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
MCR-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
MCR-02C	Informational	Yes	Misleading Error Message
SSG-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
SSG-02C	Informational	Yes	Inefficient & Outdated Square Root Implementation
SSG-03C	Informational	Yes	Repetitive Value Literal
SSS-01C	Informational	Yes	Generic Typographic Mistake
SSS-02C	Informational	Yes	Inefficient Data Structure

MerkleClaimer Static Analysis Findings

MCR-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	MerkleClaimer.sol:L53-L56

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

contracts/tokens/staking/MerkleClaimer.sol

SOL

```sol
53  constructor(address payable _stakingContract)
UpgradeableAccessControl(msg.sender) {
54      staking = SophonStaking(_stakingContract);
55      _setInitialized();
56  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The input `_stakingContract` address argument of the **`MerkleClaimer::constructor`** function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

SophonStaking Static Analysis Findings

SSG-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	**SophonStaking.sol**: • I-1: **L27** • I-2: **L55**

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
27  uint256 internal constant MAX_PERCENT = 10000;
```

Recommendation:

To properly illustrate each value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The referenced value literals have been updated in their representation to `100_00`, and `30_00` respectively in accordance with the recommendation's underscore style, addressing this exhibit.

MerkleClaimer Manual Review Findings

MCR-01M: Fixed Domain Separator Calculation

Type	Severity	Location
Logical Fault	Minor	MerkleClaimer.sol:L63-L71

Description:

The `MerkleClaimer::initialize` function will calculate the **EIP-712** domain separator once, causing future forks to utilize an incorrect domain separator.

Impact:

Should the blockchain the contract is deployed into fork in the future, the same domain separator will be used across two distinct contract implementations permitting signatures to be re-broadcast.

Example:

contracts/tokens/staking/MerkleClaimer.sol

SOL

```sol
62  function initialize(address admin) external notInitialized {
63      domainSeparator = keccak256(
64          abi.encode(
65              keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"),
66              keccak256(bytes("MerkleClaimer")),
67              keccak256(bytes("1")),
68              block.chainid,
69              address(this)
70          )
71      );
```

Example (Cont.):

```sol

    _grantRole(ADMIN_ROLE, admin);
}
```

Recommendation:

We advise the code to cache the initial `domainSeparator` as it presently does, and utilize it solely if the `block.chainid` that was used to calculate it has remained the same (i.e. no fork has occurred).

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The Sophon team opted For a non-automatic approach by introducing a function for the default admin role to recalculate the domain separator.

As it is now possible for the domain separator to be recalculated after a blockchain fork, we consider the exhibit alleviated.

MCR-02M: Improper Nonce Increments

Type	Severity	Location
Logical Fault	Medium	**MerkleClaimer.sol**: • I-1: **L97**, **L99** • I-2: **L122**, **L124**

Description:

The referenced nonce increments occur after an external native fund transfer and thus re-entrancy attack, permitting the same nonce to be re-used multiple times.

Impact:

It is presently possible to re-use the same nonce across several signatures through a re-entrancy attack when the claim is performed.

Example:

contracts/tokens/staking/MerkleClaimer.sol

SOL

```sol
97  _verifySignature(_user, _amount, _merkleIndex, _expiry, _signature);
98  _processClaim(_user, _user, _amount, _merkleProof, 0, _merkleIndex);
99  nonces[_user]++;
```

Recommendation:

We advise the nonce to be incremented right after being validated in each instance, preventing re-entrancy attacks from manifesting.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The statements in each referenced function have been re-ordered to ensure that the nonce has been incremented before the claim has been processed ensuring that signature re-use is no longer possible.

MCR-03M: Inexistent Restriction of Batch Operations

Type	Severity	Location
Logical Fault	Medium	**MerkleClaimer.sol**: • I-1: **L137-L151** • I-2: **L153-L170**

Description:

The `MerkleClaimer::batchClaim` and `MerkleClaimer::batchClaimAndStake` function implementations, in contrast to their non-batch counterparts, permit claims to occur for any user without validating the caller or any signature.

Impact:

Access control that is otherwise imposed in the `MerkleClaimer` contract is absent in its batch implementations, permitting any user's claim to be processed and to also be staked.

Example:

contracts/tokens/staking/MerkleClaimer.sol

SOL

```sol
137 function batchClaim(
138     address[] calldata users,
139     uint256[] calldata amounts,
140     uint256[] calldata merkleIndices,
141     bytes32[][] calldata merkleProofs
142 ) external whenNotPaused {
143     if (
144         users.length != amounts.length || users.length != merkleProofs.length
145             || users.length != merkleIndices.length
146     ) revert InvalidInputLengths();
```

Example (Cont.):

SOL

```sol

    for (uint256 i = 0; i < users.length; i++) {
        _processClaim(users[i], users[i], amounts[i], merkleProofs[i], 0, merkleIndices[i]);
    }
}

function batchClaimAndStake(
    address[] calldata users,
    uint256[] calldata amounts,
    uint256[] calldata stakeAmounts,
    uint256[] calldata merkleIndices,
    bytes32[][] calldata merkleProofs
) external {
    if (
        users.length != amounts.length || users.length != merkleProofs.length || users.length != stakeAmounts.length
            || users.length != merkleIndices.length
    ) {
        revert InvalidInputLengths();
    }

    for (uint256 i = 0; i < users.length; i++) {
        _processClaim(users[i], users[i], amounts[i], merkleProofs[i], stakeAmounts[i], merkleIndices[i]);
    }
}
```

Recommendation:

We advise this to be corrected, ensuring access control is consistently applied within the contract.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

Both batch function implementations were updated to properly enforce access-control, Alleviating this exhibit.

SophonStaking Manual Review Findings

SSG-01M: Discrepancy of Contract Initialization

Type	Severity	Location
Logical Fault	Minor	SophonStaking.sol:L51, L55

Description:

The initialization of the contract accepts an arbitrary `adminAddress_` to grant the `ADMIN_ROLE` to, however, the `SophonStaking::registerValidator` function it invokes will ensure the caller is the `ADMIN_ROLE`.

Impact:

The current implementation is non-standard and should be optimized or corrected.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
37  /**
38   * @notice Initializes the contract with initial parameters
39   * @param adminAddress_ The address to grant admin role to
40   * @param cooloffPeriod_ The duration in seconds that must pass after requesting
unstake
41   * @param globalFeePercent_ Initial global fee percentage (5000 = 50.00%)
42   * @dev Can only be called once via notInitialized modifier
43   */
44  function initialize(address adminAddress_, uint256 cooloffPeriod_, uint256
globalFeePercent_)
45      external
46      notInitialized
```

Example (Cont.):

```sol
{
    if (adminAddress_ == address(0)) revert ZeroAddress();
    if (globalFeePercent_ > MAX_PERCENT) revert GlobalFeeTooHigh();
    if (cooloffPeriod_ > MAX_COOLOFF_PERIOD) revert CooloffPeriodTooLong();
    _grantRole(ADMIN_ROLE, adminAddress_);
    cooloffPeriod = cooloffPeriod_;
    globalFeePercent = globalFeePercent_;

    registerValidator(DEFAULT_VALIDATOR, adminAddress_, 3000);
}
```

Recommendation:

We advise the code to grant the `ADMIN_ROLE` to the caller or the code to invoke an optimized variant of the **`SophonStaking::registerValidator`** function that does not apply the `onlyRole` modifier, either of which we consider an adequate alleviation of this exhibit.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The code was corrected to invoke an internal variant of the **`SophonStaking::registerValidator`** function addressing this exhibit.

SSG-02M: Incorrect Assumption of Staked Funds

Type	Severity	Location
Logical Fault	Medium	SophonStaking.sol:L996

Description:

The `SophonStaking::receive` function will incorrectly consider paused validators as part of the funds that rewards should be distributed to and will additionally mistake any accidentally sent funds to the contract (f.e. block rewards or `selfdestruct` operations which are still permitted in the same transaction a contract is created) as staked funds.

Impact:

Reward disbursements are presently calculated incorrectly due to an incorrect assumption about the smart contract's native balance.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
993 receive() external payable {
994     if (msg.value == 0) return;
995
996     uint256 balanceForCalc = address(this).balance - msg.value;
997     if (balanceForCalc == 0) {
998         // no one has staked anything yet
999         revert NoStakedSophon();
100
0       }
100
1
100
2       uint256 globalFeeAmount = msg.value * globalFeePercent / MAX_PERCENT;
```

Example (Cont.):

```sol
        _globalFeePerValidator += globalFeeAmount / _validatorAddresses.length;

        uint256 rewardsNotAllocated = msg.value - globalFeeAmount;

        uint256 totalStaked = _totalStaked;
        if (totalStaked != 0) {
            uint256 utilizedRewards;

            uint256 a = sqrt(balanceForCalc);
            uint256 b = _circulatingSupplySqrt;
            if (a > b) {
                utilizedRewards = rewardsNotAllocated;
            } else {
                utilizedRewards = rewardsNotAllocated * a / b;
            }
            rewardsNotAllocated -= utilizedRewards;

            // track utilized rewards per token to distribute validator fees and user rewards
            _utilizedRewardsPerToken += utilizedRewards * 1e36 / totalStaked;

            // track user rewards after removing validator fees
```

```
102
4           _globalUserRewardsUnsettled += utilizedRewards - (utilizedRewards *
_validatorProfitsRatio / 1e36);
102
5       }
102
6
102
7       // unutilized rewards
102
8       unutilizedRewards += rewardsNotAllocated;
102
9   }
```

Recommendation:

We advise the code to utilize a proper variable that is maintained and represents all staked funds in the system, preventing incorrect reward calculations in the `SophonStaking::receive` function implementation.

Alleviation (e04476671d):

The Sophon team evaluated this exhibit and clarified that they consider it desirable behavior as they wish to keep the staking and square root calculations involved in reward settlements as simple as possible.

We would like to note that this would result in a skewed `balanceForCalc` value that is higher than expected thereby resulting in a higher `utilizedRewards` evaluation and thus an incorrect increment of the rewards per token and unsettled rewards in the case that native funds have been forced into the contract without triggering any of its usual functions.

As a result, we consider the current code incorrect when it comes to native funds that have been forced into the contract.

Alleviation (564f8b9293):

The code was updated to utilize the `SophonStaking::totalStakedAmount` result, effectively representing the contract's balance sheet without relying on its dynamic `address(this).balance` result and thus alleviating this exhibit in full.

SSG-03M: Inexistent Prevention of Validator Profit Numerator Underflow

Type	Severity	Location
Mathematical Operations	Medium	SophonStaking.sol:L909, L936

Description:

The `validatorProfitsNumerator` variable updates performed in the `SophonStaking::_updateValidatorProfitsRatioWithAdd` and `SophonStaking::_updateValidatorProfitsRatioWithSub` functions can result in an underflow if two additions that truncate are performed with one subtraction that does not truncate.

Impact:

It is possible to cause the system to result in an underflow when subtracting staked amounts and thus when withdrawing due to an insecure subtraction performed for the validator profits numerator.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
894 /**
895  * @notice Internal function to update validator profits ratio when adding stake
896  * @param validator The address of the validator
897  * @param addAmount The amount being added to the stake
898  */
899 function _updateValidatorProfitsRatioWithAdd(address validator, uint256
addAmount) internal {
900     if (addAmount == 0) {
901         return;
902     }
903
```

Example (Cont.):

```sol
    uint256 totalStaked = _totalStaked;
    uint256 validatorProfitsNumerator = _validatorProfitsNumerator;

    totalStaked = totalStaked + addAmount;
    validatorProfitsNumerator =
        validatorProfitsNumerator + addAmount * validators[validator].feePercent / MAX_PERCENT;

    _validatorProfitsRatio = validatorProfitsNumerator * 1e36 / totalStaked;
    _totalStaked = totalStaked;
    _validatorProfitsNumerator = validatorProfitsNumerator;
}

/**
 * @notice Internal function to update validator profits ratio when removing stake
 * @param validator The address of the validator
 * @param subAmount The amount being subtracted from the stake
 */
function _updateValidatorProfitsRatioWithSub(address validator, uint256 subAmount) internal {
    if (subAmount == 0) {
        return;
    }

    uint256 totalStaked = _totalStaked;
    uint256 validatorProfitsNumerator = _validatorProfitsNumerator;

    if (totalStaked > subAmount) {
        totalStaked -= subAmount;
    } else {
```

Example (Cont.):

```sol
        totalStaked = 0;
    }

    if (totalStaked != 0) {
        validatorProfitsNumerator -= subAmount * validators[validator].feePercent / MAX_PERCENT;
        _validatorProfitsRatio = validatorProfitsNumerator * 1e36 / totalStaked;
    } else {
        validatorProfitsNumerator = 0;
        _validatorProfitsRatio = 0;
    }

    _totalStaked = totalStaked;
    _validatorProfitsNumerator = validatorProfitsNumerator;
}
```

Recommendation:

We advise the code to use a saturating subtraction for the `validatorProfitsNumerator` update akin to the rest of the codebase, preventing underflows from occurring and preventing a subtraction update and thus a withdrawal from the system.

Alleviation (e04476671d):

The Sophon team requested a tangible example of where this vulnerability would occur.

As detailed in the description chapter, two additions that truncate followed by a subtraction that does not truncate will result in the vulnerability manifesting.

Let us consider `validators[validator].feePercent` as `MAX_PERCENT / 2`. Introducing an `addAmount` of `1` twice will result in `addAmount * validators[validator].feePercent / MAX_PERCENT` amounting to zero in both instances.

If we follow these operations with a subtraction of `subAmount` as `2`, the evaluation of `subAmount * validators[validator].feePercent / MAX_PERCENT` will result in `1` and thus underflow.

Alleviation (564f8b9293):

The code was updated to utilize a saturating subtraction as advised, capping the subtraction result to `0` if it would diverge into the negative range.

SSG-04M: Significantly Error-Prone Mathematical Calculations

Type	Severity	Location
Logical Fault	Medium	SophonStaking.sol:L908-L909, L911, L936-L937, L961-L96

Description:

The current mechanism for calculating the pro-rata cumulative fees imposed across all validators of the system is significantly prone to rounding errors and can result in fund loss through a Denial-of-Service as already demonstrated via a dedicated exhibit.

Impact:

The current validator profit calculations that are taken into account when distributing rewards are prone to rounding errors and should be overhauled to minimize them.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
993 receive() external payable {
994     if (msg.value == 0) return;
995
996     uint256 balanceForCalc = address(this).balance - msg.value;
997     if (balanceForCalc == 0) {
998         // no one has staked anything yet
999         revert NoStakedSophon();
100
0       }
100
1
100
2       uint256 globalFeeAmount = msg.value * globalFeePercent / MAX_PERCENT;
```

Example (Cont.):

```sol
        _globalFeePerValidator += globalFeeAmount / _validatorAddresses.length;

        uint256 rewardsNotAllocated = msg.value - globalFeeAmount;

        uint256 totalStaked = _totalStaked;
        if (totalStaked != 0) {
            uint256 utilizedRewards;

            uint256 a = sqrt(balanceForCalc);
            uint256 b = _circulatingSupplySqrt;
            if (a > b) {
                utilizedRewards = rewardsNotAllocated;
            } else {
                utilizedRewards = rewardsNotAllocated * a / b;
            }
            rewardsNotAllocated -= utilizedRewards;

            // track utilized rewards per token to distribute validator fees and user rewards
            _utilizedRewardsPerToken += utilizedRewards * 1e36 / totalStaked;

            // track user rewards after removing validator fees
```

```
            _globalUserRewardsUnsettled += utilizedRewards - (utilizedRewards *
_validatorProfitsRatio / 1e36);
        }


        // unutilized rewards
        unutilizedRewards += rewardsNotAllocated;
    }
```

Recommendation:

We advise the system to be overhauled, imposing the validator profit fee solely whenever a validator's rewards are synchronized rather than on distribution thereby preventing rounding errors in cumulative validator profit ratio calculations from manifesting as Denial-of-Service attacks or similar flaws.

Alleviation (e04476671d):

The Sophon team specified that they have done substantial testing across many different configurations and that they anticipate the accumulation of rounding errors to result in dust amounts.

We do not refute this claim in the exhibit and instead specify that rounding errors can result in general misbehaviours. For example, when settling rewards the code will utilize a validator profit ratio to calculate the unsettled rewards of users.

This calculation can truncate whilst the actual validator profits may not and as illustrated in the previous exhibit this can have severe consequences for the protocol.

Given the presence of undetected truncation errors in the codebase, we strongly recommend testing to be fortified by introducing fuzz tests to the codebase that evaluate a wide variety of scenarios as well as operations to ensure that contract operates as expected across all its functions regardless of the truncations its calculations have resulted in.

Alleviation (564f8b9293):

The Sophon Network team assessed our concerns and proceeded with introducing comprehensive test suites with multiple users to confirm that the truncation errors observed solely amount to dust amounts that are acceptable for their purposes.

As the Sophon Network team has assessed the rounding behaviour outlined to behave within acceptable bounds for their business purposes, we consider this exhibit safely acknowledged.

SSG-05M: Incorrect Validator Registration Mechanism

Type	Severity	Location
Logical Fault	Major	SophonStaking.sol:L473, L858

Description:

The `SophonStaking::registerValidator` function will result in any newly introduced validator to capture the full `_globalFeePerValidator` value as `_validatorProfits` due to containing an uninitialized `_globalFeePerValidatorPaid` data entry.

Impact:

Any newly introduced validator will be able to capture rewards that would tap into the funds of other validators.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
464 function registerValidator(address validator, address profitsManager, uint256
feePercent)
465     public
466     onlyRole(ADMIN_ROLE)
467 {
468     if (validators[validator].isRegistered) revert AlreadyRegistered();
469     if (profitsManager == address(0)) revert ZeroAddress();
470     if (validator == address(0)) revert ZeroAddress();
471     if (feePercent > MAX_PERCENT) revert FeeTooHigh(feePercent, MAX_PERCENT);
472
473     settleEarnings(validator, address(0));
```

Example (Cont.):

```sol

    validators[validator] =
        ValidatorInfo({isRegistered: true, feePercent: feePercent,
profitsManager: profitsManager, isPaused: false});

    _validatorAddresses.push(validator);
    _validatorIndex[validator] = _validatorAddresses.length - 1;

    emit ValidatorRegistered(validator, profitsManager, feePercent);
}
```

Recommendation:

We advise the data entry to be initialized prior to the earning settlement of a newly introduced validator, preventing incorrect validator rewards from being disbursed.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The earning settlement function was updated to yield no new profits and no rewards if the validator has not been registered yet, alleviating this exhibit.

SSG-06M: Staked Fund Duplication Attack

Type	Severity	Location
Logical Fault	Major	SophonStaking.sol:L137-L138, L152, L156

Description:

The `SophonStaking::switchValidator` function does not prevent a validator switch to occur for the same validator.

Doing so will result in a cached `userStakedTo` data entry being written in the `userStaked` mapping, thereby permitting funds of other users to be incorrectly withdrawn.

Impact:

It is presently possible to siphon all funds from all validators by performing a low value stake, duplicating until achieving the necessary balance, and withdrawing.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
124 function switchValidator(address fromValidator, address toValidator, uint256
amount) external {
125     if (amount == 0) revert StakeIsZero();
126     if (fromValidator == address(0) || toValidator == address(0)) revert
ZeroAddress();
127     if (!validators[fromValidator].isRegistered) revert
ValidatorNotRegistered(fromValidator);
128     if (!validators[toValidator].isRegistered) revert
ValidatorNotRegistered(toValidator);
129     if (validators[toValidator].isPaused) revert ValidatorIsPaused(toValidator);
130
131     // settle earnings for both validators
132     settleEarnings(fromValidator, msg.sender);
133     settleEarnings(toValidator, msg.sender);
```

Example (Cont.):

SOL

```sol

    mapping(address => uint256) storage userStaked = _userStaked[msg.sender];

    uint256 userStakedFrom = userStaked[fromValidator];
    uint256 userStakedTo = userStaked[toValidator];

    if (userStakedFrom == 0) revert InsufficientBalance(msg.sender,
fromValidator, amount, 0);

    if (amount != type(uint256).max) {
        if (amount > userStakedFrom) {
            revert InsufficientBalance(msg.sender, fromValidator, amount,
userStakedFrom);
        }
    } else {
        // full unstake
        amount = userStakedFrom;
    }

    // update balances
    userStaked[fromValidator] = userStakedFrom - amount;
    _validatorStaked[fromValidator] -= amount;
    _updateValidatorProfitsRatioWithSub(fromValidator, amount);

    userStaked[toValidator] = userStakedTo + amount;
    _validatorStaked[toValidator] += amount;
    _updateValidatorProfitsRatioWithAdd(toValidator, amount);

    // add toValidator to user's validators if not already present
    if (userStakedTo == 0 && _userUnstaking[msg.sender][toValidator] == 0) {
```

Example (Cont.):

```sol
        _addValidatorToUser(msg.sender, toValidator);
    }

    // check if user has fully unstaked from fromValidator
    if (userStakedFrom == amount && _userUnstaking[msg.sender][fromValidator] ==
0) {
        _removeValidatorFromUser(msg.sender, fromValidator);
    }

    emit Stake(msg.sender, toValidator, amount);
    emit Unstake(msg.sender, fromValidator, amount);
}
```

Recommendation:

We advise the code to either prevent a validator switch when the `fromValidator` and `toValidator` match or to always utilize up-to-date measurements for both from and to validator balance updates.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The code was updated by applying both of our recommendations, preventing a validator switch to occur between addresses pointing to the same validator and utilizing an up-to-date measurement for the `userStakedTo` balance.

MerkleClaimer Code Style Findings

MCR-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	MerkleClaimer.sol:L250

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

contracts/tokens/staking/MerkleClaimer.sol

SOL

```sol
235 if (_stakeAmount > _amount) revert InvalidInputLengths();
236
237 bytes32 leaf = keccak256(abi.encodePacked(_merkleIndex, _user, _amount));
238 if (!MerkleProof.verify(_merkleProof, merkleRoot, leaf)) revert
InvalidMerkleProof();
239
240 isClaimed[_merkleIndex] = true;
241
242 if (_stakeAmount != 0) {
243     staking.stakeOnBehalf{value: _stakeAmount}(_user);
244     emit ClaimedAndStaked(_user, _amount, _stakeAmount, _merkleIndex);
```

Example (Cont.):

```sol
} else {
    emit Claimed(_user, _amount, _merkleIndex);
}

if (_stakeAmount != _amount) {
    (bool success,) = _receiver.call{value: _amount - _stakeAmount}("");
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The reference arithmetic operation has been safely wrapped in an `unchecked` code block as advised.

MCR-02C: Misleading Error Message

Type	Severity	Location
Code Style	Informational	MerkleClaimer.sol:L235

Description:

The referenced `error` message does not match the conditional evaluated.

Example:

contracts/tokens/staking/MerkleClaimer.sol

SOL

```sol
235 if (_stakeAmount > _amount) revert InvalidInputLengths();
```

Recommendation:

We advise this to be corrected, optimizing the code's legibility.

Alleviation (e04476671d):

The exhibit contained an incorrect highlight in its example and has since been corrected.

As such, we advise it to be revisited.

Alleviation (564f8b9293):

A proper `InvalidStakeAmount` error was introduced for the referenced line addressing this exhibit.

SophonStaking Code Style Findings

SSG-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**SophonStaking.sol**: • I-1: **L415** • I-2: **L422** • I-3: **L592** • I-4: **L930**

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
414 if (newValidatorUserRewardsUnsettled > newUserRewards) {
415     newValidatorUserRewardsUnsettled -= newUserRewards;
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

All referenced arithmetic operations have been safely wrapped in an `unchecked` code block each, addressing this exhibit.

SSG-02C: Inefficient & Outdated Square Root Implementation

Type	Severity	Location
Gas Optimization	Informational	SophonStaking.sol:L975-L987

Description:

The `SophonStaking::sqrt` implementation represents an outdated and slightly optimized variant of the original OpenZeppelin `Math::sqrt` implementation.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```sol
/**
 * @notice Calculates the square root of a number
 * @param x The number to calculate the square root of
 * @return y The square root of the input
 * @dev Uses the Babylonian method for square root approximation
 */
function sqrt(uint256 x) internal pure returns (uint256 y) {
    if (x == 0) return 0;

    // initial estimate
```

Example (Cont.):

SOL

```
979     y = x;
980     uint256 z = (y + (x / y)) >> 1;
981
982     // iterate until we reach the result
983     while (z < y) {
984         y = z;
985         z = (y + (x / y)) >> 1;
986     }
987 }
```

Recommendation:

We advise an updated variant to be utilized, optimizing the code's accuracy as well as efficiency.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The code was updated to utilize the `Math` library and specifically the `Math:sqrt` function, addressing this exhibit.

SSG-03C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	SophonStaking.sol:L863, L871, L890, L911, L937, L964, L10

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

contracts/tokens/staking/SophonStaking.sol

SOL

```
863 validatorStaked * (_utilizedRewardsPerToken -
_utilizedRewardsPerTokenPaid[validator]) / 1e36;
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The referenced value literal `1e36` has been properly relocated to a contract-level `constant` declaration labelled `expScale`, optimizing the code's legibility.

SophonStakingState Code Style Findings

SSS-01C: Generic Typographic Mistake

Type	Severity	Location
Code Style	Informational	SophonStakingState.sol:L15

Description:

The referenced line contains a typographical mistake (i.e. `private` variable without an underscore prefix, a non-`snake_case` module) or generic documentational error (i.e. copy-paste) that should be corrected.

Example:

contracts/tokens/staking/SophonStakingState.sol

SOL

```sol
15  uint256 internal withdrawalWindow; // potential future enhancement to add a withdrawal window to unstake requests
```

Recommendation:

We advise this to be done so to enhance the legibility of the codebase.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The referenced variable was properly prefixed with an underscore (_), addressing this exhibit.

SSS-02C: Inefficient Data Structure

Type	Severity	Location
Gas Optimization	Informational	SophonStakingState.sol:L69-L74

Description:

The referenced data structure is inefficient as it will occupy a full 256-bit slot for a single `bool` variable.

Example:

contracts/tokens/staking/SophonStakingState.sol

SOL

```sol
struct ValidatorInfo {
    bool isRegistered;
    uint256 feePercent;
    address profitsManager;
    bool isPaused;
}
```

Recommendation:

We advise the `isRegistered` variable to be relocated after the `isPaused` variable, permitting the `profitsManager`, `isPaused`, and `isRegistered` variables to be stored in a single 256-bit data slot.

Alleviation (e04476671d0e7ac46c28fc524a7a633c3069ab56):

The data structure was restructured as advised, optimizing its gas cost.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

- Impact (High): A core invariant of the protocol can be broken for an extended duration.
- Impact (Moderate): A non-core invariant of the protocol can be broken for an extended duration or at scale, or an otherwise major-severity issue is reduced due to hypotheticals or external factors affecting likelihood.
- Impact (Low): A non-core invariant of the protocol can be broken with reduced likelihood or impact.
- Impact (None): A code or documentation flaw whose impact does not achieve low severity, or an issue without theoretical impact; a valuable best-practice
- Likelihood (High): A flaw in the code that can be exploited trivially and is ever-present.
- Likelihood (Moderate): A flaw in the code that requires some external factors to be exploited that are likely to manifest in practice.
- Likelihood (Low): A flaw in the code that requires multiple external factors to be exploited that may manifest in practice but would be unlikely to do so.
- Likelihood (None): A flaw in the code that requires external factors proven to be impossible in a production environment, either due to mathematical constraints, operational constraints, or system-related factors (i.e. EIP-20 tokens not being re-entrant).

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this 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 any other type of advice.