OMNISCIA

February 8, 2025

SMART CONTRACT AUDIT REPORT

Sophon
Farming System


omniscia.io


info@omniscia.io


Online report: sophon-farming-system

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-farming-system

Farming System Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
2450efc999	December 16th 2024	e80ca7ce5a
161b6255b6	January 3rd 2025	5e451e7747
50cf8bb12c	February 6th 2025	d62f1260f1
29014097ea	February 8th 2025	cbae4812ec

Audit Overview

We were tasked with performing an audit of the Sophon codebase and in particular their Farming System module meant to facilitate a smooth transition from an allocation point reward system base implementation to a layer-2 oracle based allocation reward system.

The system integrates with the zkSync infrastructure to facilitate cross-chain transactions and integrates with the Stork ecosystem to evaluate oracle measurements on the Sophon network.

As outlined in the audit report, the Stork integration point was impossible to verify due to inadequate documentation on the Stork's publicly available resources and we strongly advise the Sophon team to either introduce adequate documentation themselves or to choose a different integration partner.

Over the course of the audit, we identified a significant flaw in the layer-2 implementation of the Sophon farming system which revolves around the usage of an oracle-based reward distribution mechanism which is malfunctioning and can be manipulated as it applies pool value updates and reward proportion calculations atomically.

We advise the Sophon 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 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 and have identified that certain exhibits have not been adequately dealt with. We advise the Sophon team to revisit the following exhibits: **SFG-08M**, **SFG-05M**, **SFG-07M**, **SFG-04M**, **SFL-03M**, **SFL-02M**, **SFL-01M**

Additionally, the following `informational` findings remain unaddressed and should be revisited: **MAP-02C**, **SFG-03C**, **SFG-09C**, **SFG-01S**, **SFG-10C**, **SFG-06C**, **SFG-05C**, **SFG-11C**, **SFL-01C**, **SFL-06C**, **SFL-03C**, **SFL-02C**, **SFL-07C**

Post-Audit Conclusion (50cf8bb12c)

The Sophon team provided us with a commit hash to evaluate alleviations for on certain exhibits whilst justified the absence of alleviations for others.

In detail, the `SophonFarming` contract is no longer in use per the information shared with us by the Sophon team rendering any exhibits concerning it to be considered safely acknowledged.

Our reservations in relation to the Stork integration remain, however, the Sophon team has opted to retain their integration point as they believe it is crucial in their project's operation.

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

Post-Audit Conclusion (29014097ea)

The Sophon team evaluated the updated audit report and wished to address the last exhibit in an unfinalized state, `MAP-02C`, which was considered partially alleviated.

After evaluating that it has been fully alleviated as advised, we re-iterate our previous assessment and consider all outputs of the audit report properly consumed by the Sophon team.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	28	16	0	12
Minor	4	2	0	2
Medium	7	4	0	3
Major	1	1	0	0

During the audit, we filtered and validated a total of **5 findings utilizing static analysis** tools as well as identified a total of **35 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/farming-contracts**
- Commit: 2450efc9995481a17aa9ddf3d5355a63225cbde3
- Language: Solidity
- Network: Ethereum, Sophon
- Revisions: **2450efc999**, **161b6255b6**, **50cf8bb12c**, **29014097ea**

Contracts Assessed

File	Total Finding(s)
contracts/airdrop/MerkleAirdrop.sol (MAP)	4
contracts/farm/SophonFarming.sol (SFG)	21
contracts/farm/SophonFarmingL2.sol (SFL)	15
contracts/farm/SophonFarmingState.sol (SFS)	0
contracts/proxies/UUPSProxy.sol (UUP)	0

Compilation

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

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

```bash
brownie compile
```

The `brownie` tool automatically selects Solidity version `0.8.26` based on the `pragma` statements contained within the contracts.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements are open-ended (`^0.8.26`).

We locked version `0.8.26` for our static analysis as well as optimizational review of the codebase in line with the `brownie-config.yaml` file in the codebase.

To be able to run our suite of static analysis tools, we ported the codebase to a barebones `hardhat` installation to produce the proper compilation artifacts.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the codebase's bytecode size or syntax.

Static Analysis

The execution of our static analysis toolkit identified **118 potential issues** within the codebase of which **103 were ruled out to be false positives** or negligible findings.

The remaining **15 issues** were validated and grouped and formalized into the **5 exhibits** that follow:

ID	Severity	Addressed	Title
MAP-01S	Informational	Yes	Inexistent Sanitization of Input Address
SFG-01S	Informational	Acknowledged	Illegible Numeric Value Representations
SFG-02S	Informational	Yes	Inexistent Sanitization of Input Addresses
SFL-01S	Informational	Yes	Illegible Numeric Value Representations
SFL-02S	Informational	Yes	Inexistent Sanitization of Input Address

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Sophon's farming infrastructure.

As the project at hand implements a custom farming implementation with oracle integrations and a cross-chain component, 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 a significant vulnerability** within the system's layer-2 component which could have had **severe ramifications** to its overall operation; for more information, kindly consult the relevant exhibit within the audit report as well as the audit report's summary.

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 a certain extent, however, we strongly recommend it to be expanded at certain complex points such as the Stork integration code.

A total of **35 findings** were identified over the course of the manual review of which **14 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
SFG-01M	Informational	Yes	Unknown System Integration
SFG-02M	Minor	Yes	Inexistent Enforcement of Minimum Withdrawal Blocks
SFG-03M	Minor	Acknowledged	Inexplicable Usage of Distinct Argument
SFG-04M	Minor	Acknowledged	Potentially Irrecoverable Failure Scenario
SFG-05M	Medium	Acknowledged	Incompatible Boost Multiplier Systems

SFG-06M	Medium	Yes	Incorrect & Deprecated Approval Increase
SFG-07M	Medium	Acknowledged	Incorrect Restriction of Bridge
SFG-08M	Medium	Acknowledged	Inexistent Restriction of Specific Pool Configuration
SFL-01M	Informational	Acknowledged	Potentially Dangerous Integration Point
SFL-02M	Minor	Yes	Inexistent Enforcement of Minimum Withdrawal Blocks
SFL-03M	Medium	Yes	Inexistent Sanitization of Pool Overwrite
SFL-04M	Medium	Yes	Inexistent Update of Pool State
SFL-05M	Medium	Yes	Inexistent Validation of Pool ID Existence
SFL-06M	Major	Yes	Incorrect Pool Update Mechanisms

Code Style

During the manual portion of the audit, we identified **21 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
MAP-01C	Informational	Yes	Inefficient Logic Import
MAP-02C	Informational	Yes	Inefficient `mapping` Lookups
MAP-03C	Informational	Yes	Non-Standard Usage of Library
SFG-01C	Informational	Yes	Deprecated Representation of Maximum
SFG-02C	Informational	Yes	Duplication of Restrictions
SFG-03C	Informational	Acknowledged	Generic Typographic Mistakes
SFG-04C	Informational	Yes	Incorrect Error
SFG-05C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
SFG-06C	Informational	Acknowledged	Inefficient Application of Modifier
SFG-07C	Informational	Yes	Inefficient Conditional Structures

SFG-08C	Informational	Yes	Inefficient `mapping` Lookups
SFG-09C	Informational	Acknowledged	Repetitive Value Literals
SFG-10C	Informational	Acknowledged	Sub-Optimal Evaluation of Farming Conclusion
SFG-11C	Informational	Acknowledged	Unreachable Code
SFL-01C	Informational	Nullified	Deprecated Revert Pattern
SFL-02C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
SFL-03C	Informational	Acknowledged	Inefficient Application of Modifier
SFL-04C	Informational	Yes	Inefficient Conditional Structures
SFL-05C	Informational	Yes	Inefficient `mapping` Lookups
SFL-06C	Informational	Acknowledged	Repetitive Value Literals
SFL-07C	Informational	Acknowledged	Sub-Optimal Evaluation of Farming Conclusion

MerkleAirdrop Static Analysis Findings

MAP-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	MerkleAirdrop.sol:L40-L48

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/airdrop/MerkleAirdrop.sol

SOL

```sol
function initialize(address _SF_L2) public initializer {
    SF_L2 = SophonFarmingL2(_SF_L2);

    __AccessControl_init();
    __UUPSUpgradeable_init();

    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(ADMIN_ROLE, msg.sender);
}
```

Recommendation:

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

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The input `_SF_L2` address argument of the **`MerkleAirdrop::initialize`** function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

SophonFarming Static Analysis Findings

SFG-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	SophonFarming.sol:L132, L341, L852

Description:

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

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
132 if (_pointsPerBlock < 1e18 || _pointsPerBlock > 1000e18) {
```

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 (161b6255b6):

All referenced variables except for the last `l2GasLimit` literal have been adjusted as advised, partially addressing this exhibit.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-02S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	SophonFarming.sol:L95-L105

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

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/farm/SophonFarming.sol

SOL

```sol
95  constructor(address[8] memory tokens_, uint256 _CHAINID) {
96      dai = tokens_[0];
97      sDAI = tokens_[1];
98      weth = tokens_[2];
99      stETH = tokens_[3];
100     wstETH = tokens_[4];
101     eETH = tokens_[5];
102     eETHLiquidityPool = tokens_[6];
103     weETH = tokens_[7];
104     CHAINID = _CHAINID;
```

Example (Cont.):

```sol
105 }
```

Recommendation:

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

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

All input arguments of the `SophonFarming::constructor` function are adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

SophonFarmingL2 Static Analysis Findings

SFL-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	SophonFarmingL2.sol:L349, L529

Description:

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

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```
349 if (_pointsPerBlock < 1e18 || _pointsPerBlock > 1000e18) {
```

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 (161b6255b602b141630efc36a0560b1c8235403b):

The first value literal was properly configured with an underscore whilst the second literal has been omitted from the codebase rendering this exhibit addressed.

SFL-02S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	SophonFarmingL2.sol:L89-L96

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/farm/SophonFarmingL2.sol

SOL

```sol
89  constructor(address _MERKLE, address _stork) {
90      MERKLE = _MERKLE;
91
92      if (_stork == address(0)) {
93          revert ZeroAddress();
94      }
95      stork = IStork(_stork);
96  }
```

Recommendation:

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

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The input `_MERKLE` address argument of the **`SophonFarmingL2::constructor`** function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

SophonFarming Manual Review Findings

SFG-01M: Unknown System Integration

Type	Severity	Location
Logical Fault	Informational	SophonFarming.sol:L844-L845

Description:

The `PENDLE_EXCEPTION` system appears to be improperly defined and implemented as it will subtract the user's presumable balance from the `depositAmount` to permit withdrawals on the contract, however, a brief inspection of the address literal utilized in the system indicates that it does not contain any boosted balance.

Impact:

The current integration of the `PENDLE_EXCEPTION` seems incorrect as it will subtract a value of zero inefficiently.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
843 if (_pid == BEAM_WEHT_PID) {
844     UserInfo storage user = userInfo[BEAM_WEHT_PID][PENDLE_EXCEPTION];
845     depositAmount -= user.depositAmount - user.boostAmount / boosterMultiplier;
846 }
```

Recommendation:

We advise the integration to be clarified by either omitting the boost amount related code or by updating the address to the proper contract integrated.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The `PENDLE_EXCEPTION` custom integration code has had its PID updated from the `BEAM_WEHT_PID` to the `PEPE_PID` ID which properly supports boosted balances and thus causes this integration to be valid.

To note, the actual implementation of the Pepe pool is out of scope of the audit and this exhibit relates to the local integration point.

SFG-02M: Inexistent Enforcement of Minimum Withdrawal Blocks

Type	Severity	Location
Input Sanitization	Minor	SophonFarming.sol:L314

Description:

The `SophonFarming::setEndBlock` function does not enforce a minimum `_withdrawalBlocks` value, permitting farming to end simultaneously with withdrawals.

Impact:

It is presently possible to configure the system to conclude withdrawals at the same time as farming which we consider incorrect.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
314 function setEndBlock(uint256 _endBlock, uint256 _withdrawalBlocks) external
onlyOwner {
315     if (isFarmingEnded()) {
316         revert FarmingIsEnded();
317     }
318     uint256 _endBlockForWithdrawals;
319     if (_endBlock != 0) {
320         if (getBlockNumber() > _endBlock) {
321             revert InvalidEndBlock();
322         }
323         _endBlockForWithdrawals = _endBlock + _withdrawalBlocks;
```

Example (Cont.):

```sol
    } else {
        // withdrawal blocks needs an endBlock
        _endBlockForWithdrawals = 0;
    }
    massUpdatePools();
    endBlock = _endBlock;
    endBlockForWithdrawals = _endBlockForWithdrawals;
}
```

Recommendation:

We advise a sensible minimum value to be mandated for the `_withdrawalBlocks`, preventing farming and withdrawals to end at the same time.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The code was not updated per our recommendation and the restrictions of the withdrawal mechanism were updated instead, indicating that withdrawals are meant to be disabled the moment farming concludes for pools such as the `PENDLE_EXCEPTION`.

As such, we consider this exhibit adequately addressed as the justification behind the permittance of the outlined scenario has been introduced to the codebase.

SFG-03M: Inexplicable Usage of Distinct Argument

Type	Severity	Location
Input Sanitization	Minor	SophonFarming.sol:L119, L135

Description:

The `_heldProceeds` is meant to always equal the `_boostAmount` of a pool yet is accepted as a distinct argument in the **`SophonFarmingL2::addPool`** function.

Impact:

A pool's held proceeds can presently be misconfigured due to being a distinct argument from the pool's `_boostAmount`.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
99  stETH = tokens_[3];
100     wstETH = tokens_[4];
101     eETH = tokens_[5];
102     eETHLiquidityPool = tokens_[6];
103     weETH = tokens_[7];
104     CHAINID = _CHAINID;
105 }
106
107 /**
108  * @notice Allows direct deposits of ETH for deposit to the wstETH pool
```

Example (Cont.):

```sol
 */
receive() external payable {
    if (msg.sender == weth) {
        return;
    }

    depositEth(0, PredefinedPool.wstETH);
}

/**
 * @notice Initialize the farm
 * @param wstEthAllocPoint_ wstEth alloc points
 * @param weEthAllocPoint_ weEth alloc points
 * @param sDAIAllocPoint_ sdai alloc points
 * @param _pointsPerBlock points per block
 * @param _initialPoolStartBlock start block
 * @param _boosterMultiplier booster multiplier
 */
function initialize(uint256 wstEthAllocPoint_, uint256 weEthAllocPoint_, uint256 sDAIAllocPoint_, uint256 _pointsPerBlock, uint256 _initialPoolStartBlock, uint256 _boosterMultiplier) public virtual onlyOwner {
    if (_initialized) {
        revert AlreadyInitialized();
    }

    if (_pointsPerBlock < 1e18 || _pointsPerBlock > 1000e18) {
        revert InvalidPointsPerBlock();
    }
    pointsPerBlock = _pointsPerBlock;

```

Example (Cont.):

```sol
    if (_boosterMultiplier < 1e18 || _boosterMultiplier > 10e18) {
        revert InvalidBooster();
```

Recommendation:

We advise it to be removed and the `_boostAmount` to be utilized directly, optimizing the code's gas and preventing a misconfiguration of the system.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The code remains unchanged, indicating that the Sophon team wishes to acknowledge this exhibit.

We consider this exhibit acknowledged based on the fact that **the Sophon team responsibly configures pools so as to avoid a misconfiguration**.

SFG-04M: Potentially Irrecoverable Failure Scenario

Type	Severity	Location
Logical Fault	Minor	SophonFarming.sol:L904-L939

Description:

In case of a failed bridge operation, the system permits the funds to be retrieved from the zkSync Era system and resets the pool ID's `isBridged` status yet does not re-allow withdrawals.

Impact:

The bridge failure handling mechanism permits a bridge operation to be retried but offers no recourse in case the bridge itself is malfunctioning (i.e. Sophon cannot receive transactions).

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
904 /**
905  * @notice Called by an admin if a bridge process to Sophon fails
906  * @param _pid pid of the failed bridge to revert
907  */
908 function revertFailedBridge(
909     address _l1SharedBridge,
910     uint256 _pid,
911     uint256 _chainId,
912     address _depositSender,
913     address _l1Token,
```

Example (Cont.):

```sol
    uint256 _amount,
    bytes32 _l2TxHash,
    uint256 _l2BatchNumber,
    uint256 _l2MessageIndex,
    uint16 _l2TxNumberInBatch,
    bytes32[] calldata _merkleProof) external onlyOwner {

    if (address(poolInfo[_pid].lpToken) == address(0)) {
        revert PoolDoesNotExist();
    }

    IL1SharedBridge(_l1SharedBridge).claimFailedDeposit(
        _chainId,
        _depositSender,
        _l1Token,
        _amount,
        _l2TxHash,
        _l2BatchNumber,
        _l2MessageIndex,
        _l2TxNumberInBatch,
        _merkleProof
    );

    isBridged[_pid] = false;
    emit RevertFailedBridge(_pid);
}
```

Recommendation:

We advise the system to re-allow withdrawals in such a case, preventing funds from being locked improperly.

Additionally, the system should permit another mechanism to extract boosted funds (i.e. by transmitting them to a secondary address) to ensure that it can recover from such a failure scenario.

Alleviation (161b6255b6):

The withdrawal failure recovery mechanism was not introduced to the codebase, rendering this exhibit to remain open.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-05M: Incompatible Boost Multiplier Systems

Type	Severity	Location
Logical Fault	Medium	SophonFarming.sol:L353-L362, L845

Description:

The boost multiplier system permits it to be reconfigured via the `SophonFarming::setBoosterMultiplier` function, however, such an action would cause the migration of the `BEAM_WEHT_PID` to be incorrect as the `SophonFarming::_bridgePool` performs a subtraction that assumes it has not changed.

Impact:

A re-configuration of the `boosterMultiplier` after a boost has been performed by the `PENDLE_EXCEPTION` would cause the migration of the `BEAM_WEHT_PID` to measure a lower or higher `depositAmount` than actually expected depending on whether the `boosterMultiplier` was increased or decreased.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
829 function _bridgePool(uint256 _pid, uint256 _mintValue, address _sophToken,
IBridgehub _bridge) internal {
830
831     if (!isFarmingEnded() || !isWithdrawPeriodEnded() || isBridged[_pid]) {
832         revert Unauthorized();
833     }
834
835     updatePool(_pid);
836     PoolInfo storage pool = poolInfo[_pid];
837
838     if (pool.depositAmount == 0 || address(bridge) == address(0) || pool.l2Farm
== address(0)) {
```

Example (Cont.):

```sol
        revert BridgeInvalid();
    }
    uint256 depositAmount = IERC20(pool.lpToken).balanceOf(address(this));

    if (_pid == BEAM_WEHT_PID) {
        UserInfo storage user = userInfo[BEAM_WEHT_PID][PENDLE_EXCEPTION];
        depositAmount -= user.depositAmount - user.boostAmount / boosterMultiplier;
    }
```

Recommendation:

We advise either the system to not allow the `boosterMultiplier` to be reconfigured, or the booster multiplier of the `PENDLE_EXCEPTION` user to be recorded on the first interaction and permanently used to avoid an underflow from occurring.

Alleviation (161b6255b6):

The code has not been updated to accommodate for this scenario, rendering it to remain open.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-06M: Incorrect & Deprecated Approval Increase

Type	Severity	Location
Logical Fault	Medium	SophonFarming.sol:L861

Description:

The referenced operation utilizes the deprecated `SafeERC20::safeIncreaseAllowance` function and additionally does so incorrectly as it will attempt in most cases to increment a non-zero amount by `type(uint256).max` which would fail.

Impact:

Although unlikely, the current approval increase operation is incorrect and would permanently DoS bridge operations if a non-zero allowance that is less than `depositAmount` remains between the addresses specified.

Example:

contracts/farm/SophonFarming.sol

SOL

```
860 if (pool.lpToken.allowance(address(this), _request.secondBridgeAddress) <
depositAmount) {
861     pool.lpToken.safeIncreaseAllowance(_request.secondBridgeAddress,
type(uint256).max);
862 }
```

Recommendation:

We advise the system to utilize the `SafeERC20::forceApprove` function instead, ensuring any existing approval is overwritten by the maximum possible.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The approval mechanism was updated to utilize the `SafeERC20::forceApprove` function as advised, addressing this exhibit.

SFG-07M: Incorrect Restriction of Bridge

Type	Severity	Location
Logical Fault	Medium	SophonFarming.sol:L838-L840

Description:

The `SophonFarming::_bridgePool` function will incorrectly validate that the `PoolInfo` has a non-zero `depositAmount` even though a pool permits its full deposit amount to become boosted.

Impact:

A fully boosted pool will be unable to be bridged as well as withdraw its assets, causing a total fund loss of all assets deposited to it.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
838 if (pool.depositAmount == 0 || address(bridge) == address(0) || pool.l2Farm ==
address(0)) {
839     revert BridgeInvalid();
840 }
841 uint256 depositAmount = IERC20(pool.lpToken).balanceOf(address(this));
```

Recommendation:

We advise the system to instead ensure that the `pool.amount` is not zero, ensuring that a bridging operation can be properly performed for a fully boosted pool.

Alleviation (161b6255b6):

The scenario outlined is not accommodated by the code, continuing to permit a pool to result in a permanently locked state.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-08M: Inexistent Restriction of Specific Pool Configuration

Type	Severity	Location
Logical Fault	Medium	SophonFarming.sol:L191, L193, L822, L876

Description:

The system expects the pool ID of `7` to correspond to the USDC token, however, no such restrictions are actually enforced when pools are configured in the system. Similarly, a pool ID of `4` is considered to correspond to the Beam WETH pool.

Impact:

As pools are initialized in a sequential manner, a block re-org will result in different pool IDs being set and thus may for example result in the USDC pool to not actually be deployed at an ID of `7` even if the original transaction sequence was correct, causing subsequent deployment attempts to overwrite the pool ID of `7` with an incorrect token.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
814 /**
815  * @notice Permissionless function to allow anyone to bridge during the correct
period
816  * @param _pid pid to bridge
817  * @param _mintValue _mintValue SOPH gas price
818  */
819
820 function bridgePool(uint256 _pid, uint256 _mintValue, address _sophToken)
external payable {
821     // USDC exception
822     if (_pid == 7) {
823         revert Unauthorized();
```

Example (Cont.):

```sol
    }

    _bridgePool(_pid, _mintValue, _sophToken, bridge);
}

function _bridgePool(uint256 _pid, uint256 _mintValue, address _sophToken, IBridgehub _bridge) internal {

    if (!isFarmingEnded() || !isWithdrawPeriodEnded() || isBridged[_pid]) {
        revert Unauthorized();
    }

    updatePool(_pid);
    PoolInfo storage pool = poolInfo[_pid];

    if (pool.depositAmount == 0 || address(bridge) == address(0) || pool.l2Farm == address(0)) {
        revert BridgeInvalid();
    }
    uint256 depositAmount = IERC20(pool.lpToken).balanceOf(address(this));

    if (_pid == BEAM_WEHT_PID) {
        UserInfo storage user = userInfo[BEAM_WEHT_PID][PENDLE_EXCEPTION];
        depositAmount -= user.depositAmount - user.boostAmount / boosterMultiplier;
    }

    L2TransactionRequestTwoBridgesOuter memory _request = L2TransactionRequestTwoBridgesOuter({
        chainId: CHAINID,
        mintValue: _mintValue,
        l2Value: 0,
```

Example (Cont.):

```sol
852         l2GasLimit: 2000000,
853         l2GasPerPubdataByteLimit: 800,
854         refundRecipient: address(this),
855         secondBridgeAddress: address(bridge.sharedBridge()),
856         secondBridgeValue: 0,
857         secondBridgeCalldata: abi.encode(pool.lpToken, depositAmount,
pool.l2Farm)
858     });
859
860     if (pool.lpToken.allowance(address(this), _request.secondBridgeAddress) <
depositAmount) {
861         pool.lpToken.safeIncreaseAllowance(_request.secondBridgeAddress,
type(uint256).max);
862     }
863     IERC20(_sophToken).safeTransferFrom(msg.sender, address(this), _mintValue);
864     IERC20(_sophToken).safeIncreaseAllowance(_request.secondBridgeAddress,
_mintValue);
865
866     // Actual values are pending the launch of Sophon testnet
867     _bridge.requestL2TransactionTwoBridges(_request);
868
869     isBridged[_pid] = true;
870     emit BridgePool(msg.sender, _pid, depositAmount);
871 }
872
873
874 // bridge USDC
875 function bridgeUSDC(uint256 _mintValue, address _sophToken, IBridgehub _bridge)
external onlyOwner {
876     uint256 _pid = 7;
877     if (!isFarmingEnded() || !isWithdrawPeriodEnded() || isBridged[_pid]) {
878         revert Unauthorized();
879     }
```

Example (Cont.):

```sol
880
881     // IBridgehub _bridge = IBridgehub(address(0));
882     _bridgePool(_pid, _mintValue, _sophToken, _bridge);
883
884 }
```

Recommendation:

We advise the relevant pool IDs to either be pre-configured during the contract's initialization or to be mandated as being the relevant address when they are configured dynamically, either of which we consider an adequate alleviation to this exhibit.

Alleviation (161b6255b6):

The ID based restrictions of pools are still not enforced by the system.

To note, the BEAM-WETH pool is no longer integrated by the system and the PEPE pool has been integrated instead updating the pool ID of `4` to `9`.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SophonFarmingL2 Manual Review Findings

SFL-01M: Potentially Dangerous Integration Point

Type	Severity	Location
Standard Conformity	Informational	SophonFarmingL2.sol:L527, L529, L535

Description:

The `SophonFarmingL2::updatePool` function integrates with the Stork oracle system that does not possess adequate documentation as to its oracle implementations.

While the present integration was validated with the Stork address of the Sophon testnet network, we consider this integration point volatile as there might be more security concerns that the Stark system may expose.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```
527 IStork.TemporalNumericValue memory storkValue =
stork.getTemporalNumericValueUnsafeV1(feedHash);
528
529 if (block.timestamp - (storkValue.timestampNs / 1000000000) > pv.staleSeconds) {
530     // stale price
531     pool.lastRewardBlock = getBlockNumber();
532     return;
533 }
534
535 if (storkValue.quantizedValue <= 0) {
536     // invalid price
```

Example (Cont.):

```sol
    pool.lastRewardBlock = getBlockNumber();
    return;
}
uint256 newPrice = uint256(uint192(storkValue.quantizedValue));
```

Recommendation:

We advise the system to either contain the documentation provided by the Stork system to integrate with it, or to select a different integration partner. At its current state, the integration cannot be validated to the extent it should due to inadequate documentation on the Stork ecosystem.

Alleviation (161b6255b6):

While the overall integration has been refactored to include a `PriceFeeds` implementation that continues to integrate with Stork, we still do not possess adequate documentation from either team rendering this exhibit to remain in the codebase.

Alleviation (50cf8bb12c):

After we pointed out an error in the Stork indicative implementation that was shared with us, we were put in contact with the Stork team and discussed a few of the implementation choices that they have taken.

The Sophon team has decided that they wish to continue their integration with Stork, and we consider this exhibit to be considered acknowledged.

SFL-02M: Inexistent Enforcement of Minimum Withdrawal Blocks

Type	Severity	Location
Input Sanitization	Minor	SophonFarmingL2.sol:L331

Description:

The `SophonFarmingL2::setEndBlock` function does not enforce a minimum `_withdrawalBlocks` value, permitting farming to end simultaneously with withdrawals.

Impact:

It is presently possible to configure the system to conclude withdrawals at the same time as farming which we consider incorrect.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
317 /**
318  * @notice Set the end block of the farm
319  * @param _endBlock the end block
320  * @param _withdrawalBlocks the last block that withdrawals are allowed
321  */
322 function setEndBlock(uint256 _endBlock, uint256 _withdrawalBlocks) external
onlyOwner {
323     if (isFarmingEnded()) {
324         revert FarmingIsEnded();
325     }
326     uint256 _endBlockForWithdrawals;
```

Example (Cont.):

```sol
    if (_endBlock != 0) {
        if (getBlockNumber() > _endBlock) {
            revert InvalidEndBlock();
        }
        _endBlockForWithdrawals = _endBlock + _withdrawalBlocks;
    } else {
        // withdrawal blocks needs an endBlock
        _endBlockForWithdrawals = 0;
    }
    massUpdatePools();
    endBlock = _endBlock;
    endBlockForWithdrawals = _endBlockForWithdrawals;
}
```

Recommendation:

We advise a sensible minimum value to be mandated for the `_withdrawalBlocks`, preventing farming and withdrawals to end at the same time.

Alleviation (161b6255b6):

The restriction continues to remain unimposed and the codebase has not been updated in relation to this exhibit rendering it to remain open.

Alleviation (50cf8bb12c):

The equivalent of one day in blocks was configured as the minimum enforced for withdrawals, alleviating this exhibit in full.

SFL-03M: Inexistent Sanitization of Pool Overwrite

Type	Severity	Location
Input Sanitization	Medium	SophonFarmingL2.sol:L115-L126, L129

Description:

The `SophonFarmingL2::addPool` function permits a pool already introduced to the system to be overwritten, resulting in significant issues within the system's accounting.

Impact:

A pool overwrite can significantly compromise the system's reward and balance accounting as rewards are overwritten to an `accPointsPerShare` of `0` regardless of their previous value.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
99  function addPool(
100     uint256 _pid,
101     IERC20 _lpToken,
102     address _l2Farm,
103     uint256 _amount,
104     uint256 _boostAmount,
105     uint256 _depositAmount,
106     uint256 _allocPoint,
107     uint256 _lastRewardBlock,
108     uint256 _accPointsPerShare,
```

Example (Cont.):

```sol
    uint256 _totalRewards,
    string memory _description,
    uint256 _heldProceeds
) public onlyOwner {
    require(_amount == _boostAmount + _depositAmount, "balances don't match");

    PoolInfo memory pool = PoolInfo({
        lpToken: _lpToken,
        l2Farm: _l2Farm,
        amount: _amount,
        boostAmount: _boostAmount,
        depositAmount: _depositAmount,
        allocPoint: 0,
        lastRewardBlock: _lastRewardBlock,
        accPointsPerShare: 0,
        totalRewards: _totalRewards,
        description: _description
    });

    if (_pid < poolInfo.length) {
        poolInfo[_pid] = pool;
    } else if (_pid == poolInfo.length) {
        poolInfo.push(pool);
    } else {
        revert("wrong pid");
    }
    heldProceeds[_pid] = _heldProceeds;
    poolExists[address(_lpToken)] = true;
```

Example (Cont.):

```sol
137     // require(IERC20(_lpToken).balanceOf(address(this)) >= _amount, "balances
don't match");
138 }
```

Recommendation:

We advise such an operation to be restricted, for example by ensuring that the relevant amounts exceed their previously configured ones, that the `accPointsPerShare` value is not reset, that the `_lastRewardBlock` is properly configured, and in general that the overwrite operation is securely performed.

Alleviation (161b6255b6):

A single restriction was introduced that solely ensures the `lpToken` remains the same, rendering this exhibit partially addressed as the alleviation is inadequate.

Alleviation (50cf8bb12c):

The **`SophonFarmingL2::addPool`** function has been removed addressing this exhibit as a result.

SFL-04M: Inexistent Update of Pool State

Type	Severity	Location
Logical Fault	Medium	SophonFarmingL2.sol:L421-L432

Description:

The `SophonFarmingL2::setPriceFeedData` function permits the `emissionsMultiplier` of a pool to be reconfigured yet does not update it before performing such an operation, causing the multiplier to retroactively apply.

Impact:

A price feed data update that reconfigures the `emissionsMultiplier` will retroactively apply which is incorrect.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
// zero hash allowed; blocks updates to the pool
// zero stale seconds means no change
function setPriceFeedData(uint256 _pid, bytes32 _newHash, uint256 _newStaleSeconds, uint256 _emissionsMultiplier) external onlyOwner {
    PoolValue storage pv = poolValue[_pid];
    if (_newHash == pv.feedHash) {
        revert DuplicatePriceFeed();
    }

    pv.feedHash = _newHash;
    pv.staleSeconds = _newStaleSeconds;
```

Example (Cont.):

```sol
    pv.emissionsMultiplier = _emissionsMultiplier;

    emit SetPriceFeedData(_newHash, _newStaleSeconds);
}
```

Recommendation:

We advise the system to ensure all pools are updated before updating the price feed data of a pool, preventing inconsistencies in the system's accounting.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

All pools are now updated prior to the re-configuration of an emission multiplier ensuring that the adjustment does not apply retroactively.

SFL-05M: Inexistent Validation of Pool ID Existence

Type	Severity	Location
Logical Fault	Medium	SophonFarmingL2.sol:L144, L145

Description:

The `SophonFarmingL2::updateUserInfo` function does not validate that the `_pid` for which the `MerkleAirdrop` claim has been processed exists, permitting a user to update their information on an inexistent pool.

Impact:

A user claiming their corresponding stake in an uninitialized pool would result in their balances being overwritten, their rewards misallocated, and generally introduce an inaccuracy in the system's accounting.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
140 function updateUserInfo(address _user, uint256 _pid, UserInfo memory
_userFromClaim) public {
141     if (msg.sender != MERKLE) revert OnlyMerkle();
142     require(_userFromClaim.amount == _userFromClaim.boostAmount +
_userFromClaim.depositAmount, "balances don't match");
143
144     PoolInfo storage pool = poolInfo[_pid];
145     UserInfo storage user = userInfo[_pid][_user];
146     massUpdatePools(true);
147
148     uint256 userAmount = user.amount;
149
```

Example (Cont.):

```sol
    user.rewardSettled =
        user.amount *
        pool.accPointsPerShare /
        1e18 +
        user.rewardSettled -
        user.rewardDebt;

    // _userFromClaim.rewardDebt is ignored since user.rewardSettled is already settled
    user.rewardSettled = user.rewardSettled + _userFromClaim.rewardSettled;

    user.boostAmount = user.boostAmount + _userFromClaim.boostAmount;
    pool.boostAmount = pool.boostAmount + _userFromClaim.boostAmount;

    user.depositAmount = user.depositAmount + _userFromClaim.depositAmount;
    pool.depositAmount = pool.depositAmount + _userFromClaim.depositAmount;

    user.amount = user.amount + _userFromClaim.amount;
    pool.amount = pool.amount + _userFromClaim.amount;

    user.rewardDebt = user.amount *
        pool.accPointsPerShare /
        1e18;
}
```

Recommendation:

We advise the system to ensure that the pool ID exists by ensuring that the `_pid` is less than the `poolInfo.length`.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The pool ID for which user information is updated is now sanitized, ensuring a proper pool update occurs from the `MERKLE` implementation.

SFL-06M: Incorrect Pool Update Mechanisms

Type	Severity	Location
Logical Fault	Major	SophonFarmingL2.sol:L497-L499, L506, L527, L542, L550, L551, L555-L559

Description:

The system utilizes a novel oracle-based value evaluation mechanism to assess how many rewards emitted per interaction should be distributed to a particular pool.

A caveat of this mechanism is that it relies on a synchronized value evaluation of all active pools in the system which is presently not adhered to, permitting reward allocations to be trivially manipulated via updates of singular pools via the `SophonFarmingL2::updatePool` functions.

Furthermore, the `SophonFarmingL2::massUpdatePools` implementations are incorrect as well given that they sequentially update pools one-by-one instead of updating the values of all pools before assessing the reward due for each pool.

Impact:

All rewards calculated in the `SophonFarmingL2` system are presently incorrect and can be manipulated as the system will update oracle-based evaluations of the pools in a sequential manner, undervaluing or overvaluing the proportion of the currently-iterated pool update in relation to the upcoming pool updates.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
501 /**
502  * @notice Updating accounting of a single pool
503  * @param _pid pid to update
504  * @param _silent emit event if false
505  */
506 function updatePool(uint256 _pid, bool _silent) public {
507     PoolInfo storage pool = poolInfo[_pid];
508     if (getBlockNumber() <= pool.lastRewardBlock) {
509         return;
510     }
```

Example (Cont.):

```sol
511     uint256 lpSupply = pool.amount;
512     uint256 _pointsPerBlock = pointsPerBlock;
513     if (lpSupply == 0 || _pointsPerBlock == 0) {
514         pool.lastRewardBlock = getBlockNumber();
515         return;
516     }
517 
518     /* START Update Pool Weighting Block */
519     PoolValue storage pv = poolValue[_pid];
520     bytes32 feedHash = pv.feedHash;
521     if (feedHash == 0) {
522         //revert PriceFeedNotSet();
523         pool.lastRewardBlock = getBlockNumber();
524         return;
525     }
526 
527     IStork.TemporalNumericValue memory storkValue =
stork.getTemporalNumericValueUnsafeV1(feedHash);
528 
529     if (block.timestamp - (storkValue.timestampNs / 1000000000) >
pv.staleSeconds) {
530         // stale price
531         pool.lastRewardBlock = getBlockNumber();
532         return;
533     }
534 
535     if (storkValue.quantizedValue <= 0) {
536         // invalid price
537         pool.lastRewardBlock = getBlockNumber();
538         return;
```

Example (Cont.):

```sol
    }
    uint256 newPrice = uint256(uint192(storkValue.quantizedValue));

    uint256 newValue = lpSupply * newPrice / 1e18;
    newValue = newValue * pv.emissionsMultiplier / 1e18;
    if (newValue == 0) {
        //revert InvalidValue(newValue);
        pool.lastRewardBlock = getBlockNumber();
        return;
    }

    totalValue = totalValue - pv.lastValue + newValue;
    pv.lastValue = newValue;
    /* END Update Pool Weighting Block */

    uint256 blockMultiplier = _getBlockMultiplier(pool.lastRewardBlock,
getBlockNumber());
    uint256 pointReward =
        blockMultiplier *
        _pointsPerBlock *
        newValue /
        totalValue;

    pool.totalRewards = pool.totalRewards + pointReward / 1e18;

    pool.accPointsPerShare = pointReward /
        lpSupply +
        pool.accPointsPerShare;

```

Example (Cont.):

```sol
    pool.lastRewardBlock = getBlockNumber();

    if (!_silent) {
        emit PoolUpdated(_pid);
    }
}
```

Recommendation:

We advise the system to be refactored, no longer permitting singular pool updates to be performed.

As a second remediation step, the system should update all `lastValue` entries **before updating the `pool` entries themselves** to ensure that the system is in a synchronized value state so as to address the misattribution of rewards.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The singular publicly accessible `SophonFarmingL2::updatePool` mechanism has been omitted from the codebase and the `SophonFarmingL2::massUpdatePools` function was refactored as advised, updating all `lastValue` entries and disbursing rewards after they have been synchronized.

MerkleAirdrop Code Style Findings

MAP-01C: Inefficient Logic Import

Type	Severity	Location
Code Style	Informational	MerkleAirdrop.sol:L10

Description:

The `SophonFarmingL2` imported to the codebase is utilized as an `interface` rather than an implementation.

Example:

contracts/airdrop/MerkleAirdrop.sol

SOL

```sol
10  import "contracts/farm/SophonFarmingL2.sol";
11
12  contract MerkleAirdrop is Initializable, AccessControlUpgradeable,
UUPSUpgradeable {
13      bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
14      SophonFarmingL2 public SF_L2;
15      bytes32 public merkleRoot;
```

Recommendation:

We advise an `interface` to be properly declared for it and imported to the `MerkleAirdrop` codebase, optimizing its syntax as well as bytecode.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

A proper `interface` was declared for the `SophonFarmingL2` contract (`ISophonFarming`) and is now imported by the `MerkleAirdrop` contract addressing this exhibit in full.

MAP-02C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	MerkleAirdrop.sol:L110, L117

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/airdrop/MerkleAirdrop.sol

SOL

```sol
110 if (hasClaimed[_user][_pid]) revert AlreadyClaimed();
111
112 // Verify the Merkle proof.
113 bytes32 leaf = keccak256(abi.encodePacked(_user, _pid, _userInfo.amount,
_userInfo.boostAmount, _userInfo.depositAmount, _userInfo.rewardSettled,
_userInfo.rewardDebt));
114 if (!MerkleProof.verify(_merkleProof, merkleRoot, leaf)) revert
InvalidMerkleProof();
115
116 // Mark it claimed and transfer the tokens.
117 hasClaimed[_user][_pid] = true;
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (161b6255b6):

The optimization has been misapplied as the `bool` value is stored to memory thus increasing gas costs.

Our recommendation was to cache the `hasClaimed[_user]` lookup to a **local** `mapping(uint256 => bool) storage` variable that is supported by the Solidity syntax and would optimize the code's gas cost.

Alleviation (29014097ea):

The Sophon team revisited this exhibit and proceeded to fully remediate it as advised, optimizing the code's gas cost.

MAP-03C: Non-Standard Usage of Library

Type	Severity	Location
Code Style	Informational	MerkleAirdrop.sol:L129

Description:

The `SafeERC20` library is meant to be utilized via the `using SafeERC20 for IERC20` syntax as exposed by the `IERC20` variable directly, however, the **`MerkleAirdrop::rescue`** function invokes the `library` directly.

Example:

contracts/airdrop/MerkleAirdrop.sol

SOL

```sol
128 function rescue(IERC20 token, address to) external onlyRole(ADMIN_ROLE) {
129     SafeERC20.safeTransfer(token, to, token.balanceOf(address(this)));
130 }
```

Recommendation:

We advise the library to be utilized properly via the appropriate syntax, optimizing the code's legibility and standardization.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The `SafeERC20` library is properly utilized as exposed by the `IERC20` data type, addressing this exhibit.

SophonFarming Code Style Findings

SFG-01C: Deprecated Representation of Maximum

Type	Severity	Location
Code Style	Informational	SophonFarming.sol:L151, L155, L159

Description:

The referenced statements will represent the maximum `uint256` possible via `2**256 - 1` which is a calculation that would normally overflow in Solidity.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
148 // sDAI
149 typeToId[PredefinedPool.sDAI] = add(sDAIAllocPoint_, sDAI, "sDAI",
_initialPoolStartBlock, 0);
150 IERC20(dai).approve(sDAI, 2**256-1);
151
152 // wstETH
153 typeToId[PredefinedPool.wstETH] = add(wstEthAllocPoint_, wstETH, "wstETH",
_initialPoolStartBlock, 0);
154 IERC20(stETH).approve(wstETH, 2**256-1);
155
156 // weETH
157 typeToId[PredefinedPool.weETH] = add(weEthAllocPoint_, weETH, "weETH",
_initialPoolStartBlock, 0);
```

Example (Cont.):

```sol
158 IERC20(eETH).approve(weETH, 2**256-1);
```

Recommendation:

We advise the code to utilize the `type(uint256).max` syntax, optimizing the code's legibility as well as standardization.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

All representations of maximum have been updated to `type(uint256).max` as advised.

SFG-02C: Duplication of Restrictions

Type	Severity	Location
Gas Optimization	Informational	SophonFarming.sol:L831-L833, L877-L879

Description:

The referenced restrictions are duplicated across the `SophonFarming::_bridgePool` and `SophonFarming::bridgeUSDC` function implementations.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
829 function _bridgePool(uint256 _pid, uint256 _mintValue, address _sophToken,
IBridgehub _bridge) internal {
830
831     if (!isFarmingEnded() || !isWithdrawPeriodEnded() || isBridged[_pid]) {
832         revert Unauthorized();
833     }
834
835     updatePool(_pid);
836     PoolInfo storage pool = poolInfo[_pid];
837
838     if (pool.depositAmount == 0 || address(bridge) == address(0) || pool.l2Farm
== address(0)) {
```

Example (Cont.):

```sol
            revert BridgeInvalid();
        }
        uint256 depositAmount = IERC20(pool.lpToken).balanceOf(address(this));

        if (_pid == BEAM_WEHT_PID) {
            UserInfo storage user = userInfo[BEAM_WEHT_PID][PENDLE_EXCEPTION];
            depositAmount -= user.depositAmount - user.boostAmount / boosterMultiplier;
        }

        L2TransactionRequestTwoBridgesOuter memory _request = L2TransactionRequestTwoBridgesOuter({
            chainId: CHAINID,
            mintValue: _mintValue,
            l2Value: 0,
            l2GasLimit: 2000000,
            l2GasPerPubdataByteLimit: 800,
            refundRecipient: address(this),
            secondBridgeAddress: address(bridge.sharedBridge()),
            secondBridgeValue: 0,
            secondBridgeCalldata: abi.encode(pool.lpToken, depositAmount, pool.l2Farm)
        });

        if (pool.lpToken.allowance(address(this), _request.secondBridgeAddress) < depositAmount) {
            pool.lpToken.safeIncreaseAllowance(_request.secondBridgeAddress, type(uint256).max);
        }
        IERC20(_sophToken).safeTransferFrom(msg.sender, address(this), _mintValue);
        IERC20(_sophToken).safeIncreaseAllowance(_request.secondBridgeAddress, _mintValue);

        // Actual values are pending the launch of Sophon testnet
```

Example (Cont.):

```sol
    _bridge.requestL2TransactionTwoBridges(_request);

    isBridged[_pid] = true;
    emit BridgePool(msg.sender, _pid, depositAmount);
}


// bridge USDC
function bridgeUSDC(uint256 _mintValue, address _sophToken, IBridgehub _bridge) external onlyOwner {
    uint256 _pid = 7;
    if (!isFarmingEnded() || !isWithdrawPeriodEnded() || isBridged[_pid]) {
        revert Unauthorized();
    }

    // IBridgehub _bridge = IBridgehub(address(0));
    _bridgePool(_pid, _mintValue, _sophToken, _bridge);

}
```

Recommendation:

We advise the restrictions to be omitted from the `SophonFarming::bridgeUSDC` variant, optimizing the code's gas cost.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The `SophonFarming::bridgeUSDC` restrictions have been omitted as advised, optimizing the code's gas cost.

SFG-03C: Generic Typographic Mistakes

Type	Severity	Location
Code Style	Informational	**SophonFarming.sol**: • I-1: **L87** • I-2: **L1013** • I-3: **L1025** • I-4: **L1037** • I-5: **L1048** • I-6: **L1060** • I-7: **L1071**

Description:

The referenced lines contain typographical mistakes (i.e. `private` variable without an underscore prefix) or generic documentational errors (i.e. copy-paste) that should be corrected.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
87  uint256 internal constant BEAM_WEHT_PID = 4;
```

Recommendation:

We advise them to be corrected enhancing the legibility of the codebase.

Alleviation (161b6255b6):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-04C: Incorrect Error

Type	Severity	Location
Code Style	Informational	SophonFarming.sol:L893

Description:

The `SophonFarming::setL2Farm` function will yield an incorrect `PoolExists` error if a pool exists.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
function setL2Farm(uint256 _pid, address _l2Farm) external onlyOwner {
    if (_pid >= poolInfo.length) {
        revert PoolExists();
    }

    if (_l2Farm == address(0)) {
        revert ZeroAddress();
    }

    poolInfo[_pid].l2Farm = _l2Farm;
```

Example (Cont.):

```sol
901 }
```

Recommendation:

We advise the `PoolDoesNotExist` error to be yielded instead, properly representing the error that was identified.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The referenced `error` has been replaced with the correct `PoolDoesNotExist` declaration.

SFG-05C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**SophonFarming.sol**: • I-1: **L376** • I-2: **L668** • I-3: **L729** • I-4: **L796** • I-5: **L986**

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/farm/SophonFarming.sol

SOL

```sol
375 if (_to > _from) {
376     return (_to - _from) * 1e18;
377 } else {
```

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 (161b6255b6):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-06C: Inefficient Application of Modifier

Type	Severity	Location
Gas Optimization	Informational	SophonFarming.sol:L171, L222, L337

Description:

The `onlyOwner` modifier is redundantly applied twice in both the **`SophonFarming::add`** and **`SophonFarming::set`** functions whenever a `_newPointsPerBlock` has been configured as the **`SophonFarming::setPointsPerBlock`** function already applies it.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
222 function set(uint256 _pid, uint256 _allocPoint, uint256 _poolStartBlock, uint256 _newPointsPerBlock) external onlyOwner {
223     if (isFarmingEnded()) {
224         revert FarmingIsEnded();
225     }
226 
227     if (_newPointsPerBlock != 0) {
228         setPointsPerBlock(_newPointsPerBlock);
229     } else {
230         massUpdatePools();
231     }
```

Recommendation:

We advise the `SophonFarming::setPointsPerBlock` function to invoke an internal underscore-prefixed variant of it that does not apply this modifier, and the `SophonFarming::add` and `SophonFarming::set` functions to invoke this variant as well optimizing their gas cost.

Alleviation (161b6255b6):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-07C: Inefficient Conditional Structures

Type	Severity	Location
Gas Optimization	Informational	**SophonFarming.sol**: • I-1: **L264-L268** • I-2: **L277-L281**

Description:

The referenced `if-else` clauses will evaluate a conditional and yield `true` if it evaluates to that value.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
262 function isFarmingEnded() public view returns (bool) {
263     uint256 _endBlock = endBlock;
264     if (_endBlock != 0 && getBlockNumber() > _endBlock) {
265         return true;
266     } else {
267         return false;
268     }
269 }
```

Recommendation:

We advise the conditionals to be replaced by direct `return` statements that yield the conditional evaluated itself, optimizing the code's syntax as well as gas cost.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

Both conditional structures have been updated as advised, yielding the conditional directly.

SFG-08C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	**SophonFarming.sol**: • I-1: **L303**, **L306** • I-2: **L390**

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/farm/SophonFarming.sol

SOL

```
299 function setL2FarmForPool(uint256 _pid, address _l2Farm) external onlyOwner {
300     if (_l2Farm == address(0)) {
301         revert ZeroAddress();
302     }
303     if (address(poolInfo[_pid].lpToken) == address(0)) {
304         revert PoolDoesNotExist();
305     }
306     poolInfo[_pid].l2Farm = _l2Farm;
307 }
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The mapping lookups have been optimized as advised, greatly reducing each code segment's gas cost.

SFG-09C: Repetitive Value Literals

Type	Severity	Location
Code Style	Informational	SophonFarming.sol:L132, L137, L341, L354, L376, L406, L433, L484, L675, L72

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

contracts/farm/SophonFarming.sol

SOL

```
132 if (_pointsPerBlock < 1e18 || _pointsPerBlock > 1000e18) {
```

Recommendation:

We advise each to be set to its dedicated `constant` variable instead, optimizing the legibility of the codebase.

In case some of the `constant` declarations have already been introduced, we advise them to be properly re-used across the code.

Alleviation (161b6255b6):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-10C: Sub-Optimal Evaluation of Farming Conclusion

Type	Severity	Location
Gas Optimization	Informational	**SophonFarming.sol**: • I-1: **L178-L180**, **L183** • I-2: **L223-L225**, **L228**

Description:

The `SophonFarming::add` and `SophonFarming::end` function implementations will inefficiently evaluate whether farming has ended when simultaneously configuring a new point per block as the `SophonFarming::setPointsPerBlock` function already evaluates this restriction.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
223 if (isFarmingEnded()) {
224     revert FarmingIsEnded();
225 }
226
227 if (_newPointsPerBlock != 0) {
228     setPointsPerBlock(_newPointsPerBlock);
229 } else {
230     massUpdatePools();
231 }
```

Recommendation:

We advise the restriction to be relocated in the `else` block that mass updates pools, optimizing the code's gas cost whenever a `_newPointsPerBlock` value has been configured.

Alleviation (161b6255b6):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

SFG-11C: Unreachable Code

Type	Severity	Location
Gas Optimization	Informational	SophonFarming.sol:L624-L626

Description:

The referenced `if-else-if` chain will evaluate all possible states of an `enum` and introduce an `else` clause after they have been evaluated to `revert`.

Solidity does not permit `enum` values to exceed their explicitly declared range rendering the `else` block unreachable.

Example:

contracts/farm/SophonFarming.sol

SOL

```sol
618 if (_predefinedPool == PredefinedPool.sDAI) {
619     _finalAmount = _daiTOsDai(_amount);
620 } else if (_predefinedPool == PredefinedPool.wstETH) {
621     _finalAmount = _stEthTOwstEth(_amount);
622 } else if (_predefinedPool == PredefinedPool.weETH) {
623     _finalAmount = _eethTOweEth(_amount);
624 } else {
625     revert InvalidDeposit();
626 }
```

Recommendation:

We advise the code to omit the `else` block and to replace the last `weETH` condition with an `else` clause, optimizing the code's gas cost.

Alleviation (161b6255b6):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (50cf8bb12c):

After discussions with the Sophon team, we concluded that the contract is no longer in use and thus any finding in relation to it can be safely acknowledged.

finding in relation to it can be safely acknowledged.

SophonFarmingL2 Code Style Findings

SFL-01C: Deprecated Revert Pattern

Type	Severity	Location
Code Style	Informational	SophonFarmingL2.sol:L133

Description:

The referenced `revert` statement will revert with a string literal which has been deprecated.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
133 revert("wrong pid");
```

Recommendation:

We advise a custom `error` declaration to be introduced that is utilized in the `revert` statement, optimizing the code's legibility as well as standardization.

Alleviation (161b6255b6):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Alleviation (50cf8bb12c):

The relevant code is no longer part of the codebase rendering this exhibit nullified.

SFL-02C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**SophonFarmingL2.sol**: • I-1: **L384** • I-2: **L624** • I-3: **L685** • I-4: **L752** • I-5: **L814**

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/farm/SophonFarmingL2.sol

SOL

```sol
383 if (_to > _from) {
384     return (_to - _from) * 1e18;
```

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 (161b6255b602b141630efc36a0560b1c8235403b):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

SFL-03C: Inefficient Application of Modifier

Type	Severity	Location
Gas Optimization	Informational	**SophonFarmingL2.sol**: • I-1: **L185**, **L204** • I-2: **L253**, **L259**

Description:

The `onlyOwner` modifier is redundantly applied twice in both the **`SophonFarmingL2::add`** and **`SophonFarmingL2::set`** functions whenever a `_newPointsPerBlock` has been configured as the **`SophonFarmingL2::setPointsPerBlock`** function already applies it.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
185 function add(address _lpToken, bytes32 _priceFeedHash, uint256 _staleSeconds,
uint256 _emissionsMultiplier, string memory _description, uint256 _poolStartBlock,
uint256 _newPointsPerBlock) public onlyOwner returns (uint256) {
186     if (_lpToken == address(0)) {
187         revert ZeroAddress();
188     }
189     if (poolExists[_lpToken]) {
190         revert PoolExists();
191     }
192     if (isFarmingEnded()) {
193         revert FarmingIsEnded();
194     }
```

Example (Cont.):

```sol
195
196     if (_priceFeedHash == 0) {
197         revert PriceFeedNotSet();
198     }
199     if (_staleSeconds == 0) {
200         revert InvalidStaleSeconds();
201     }
202
203     if (_newPointsPerBlock != 0) {
204         setPointsPerBlock(_newPointsPerBlock);
205     } else {
206         massUpdatePools();
207     }
```

Recommendation:

We advise the `SophonFarmingL2::setPointsPerBlock` function to invoke an internal underscore-prefixed variant of it that does not apply this modifier, and the `SophonFarmingL2::add` and `SophonFarmingL2::set` functions to invoke this variant as well optimizing their gas cost.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

SFL-04C: Inefficient Conditional Structures

Type	Severity	Location
Gas Optimization	Informational	**SophonFarmingL2.sol**: • I-1: **L297-L301** • I-2: **L310-L314**

Description:

The referenced `if-else` clauses will evaluate a conditional and yield `true` if it evaluates to that value.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
295 function isFarmingEnded() public view returns (bool) {
296     uint256 _endBlock = endBlock;
297     if (_endBlock != 0 && getBlockNumber() > _endBlock) {
298         return true;
299     } else {
300         return false;
301     }
302 }
```

Recommendation:

We advise the conditionals to be replaced by direct `return` statements that yield the conditional evaluated itself, optimizing the code's syntax as well as gas cost.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

Both conditional structures have been updated as advised, yielding the conditional directly.

SFL-05C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	SophonFarmingL2.sol:L398

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```sol
397 for(uint i = 0; i < _users.length; i++) {
398     whitelist[_userAdmin][_users[i]] = _isInWhitelist;
399 }
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The `mapping` optimization has been applied as advised, optimizing the code's gas cost.

SFL-06C: Repetitive Value Literals

Type	Severity	Location
Code Style	Informational	SophonFarmingL2.sol:L153, L171, L232, L349, L362, L384, L414, L457, L529, L

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```
153 1e18 +
```

Recommendation:

We advise each to be set to its dedicated `constant` variable instead, optimizing the legibility of the codebase.

In case some of the `constant` declarations have already been introduced, we advise them to be properly re-used across the code.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

SFL-07C: Sub-Optimal Evaluation of Farming Conclusion

Type	Severity	Location
Gas Optimization	Informational	**SophonFarmingL2.sol**: • I-1: **L192**, **L204** • I-2: **L254**, **L259**

Description:

The `SophonFarmingL2::add` and `SophonFarmingL2::end` function implementations will inefficiently evaluate whether farming has ended when simultaneously configuring a new point per block as the `SophonFarmingL2::setPointsPerBlock` function already evaluates this restriction.

Example:

contracts/farm/SophonFarmingL2.sol

SOL

```
253 function set(uint256 _pid, uint256 _emissionsMultiplier, uint256 _poolStartBlock,
uint256 _newPointsPerBlock) external onlyOwner {
254     if (isFarmingEnded()) {
255         revert FarmingIsEnded();
256     }
257
258     if (_newPointsPerBlock != 0) {
259         setPointsPerBlock(_newPointsPerBlock);
260     } else {
261         massUpdatePools();
262     }
```

Recommendation:

We advise the restriction to be relocated in the `else` block that mass updates pools, optimizing the code's gas cost whenever a `_newPointsPerBlock` value has been configured.

Alleviation (161b6255b602b141630efc36a0560b1c8235403b):

The Sophon team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

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.