Quantstamp

Sophon Farming Program

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Cross Chain Farming
Timeline	2024-05-15 through 2024-05-22
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• https://github.com/sophon-org/farming-contracts #a4d4c0b
Auditors	• Julio Aguilar Auditing Engineer • Ruben Koch Senior Auditing Engineer

Documentation quality	Low
Test quality	High
Total Findings	11 Fixed: 8 Acknowledged: 3
High severity findings	1 Fixed: 1
Medium severity findings	3 Fixed: 3
Low severity findings	2 Fixed: 2
Undetermined severity findings	0
Informational findings	5 Fixed: 2 Acknowledged: 3

Summary of Findings

In this audit, we reviewed the `SophonFarming` and `SophonFarmingState` contracts. The contracts are a fork of the SushiSwap `MasterChefV2` contract that has been significantly modified to not distribute pending rewards, but to track them internally as they accrue. These accrued rewards will eventually be the basis for an airdrop. Furthermore, a boost mechanic has been added, with which users can essentially forfeit part of their deposit for an increased point multiplier on the forfeited amount, creating a separate so-called boosted amount. The boost multiplier and other variables are set by a trusted owner (for more information, see SOP-11). This boosted amount will continue to earn the user points independently of the remaining accessible deposit balance of the user.

Eventually, the contracts will contain a permissionless bridging mechanic, where after the farming and an optional withdrawal period have ended, the LP tokens of the pools will be able to get bridged to the Sophon blockchain, a hyper chain on ZkSync, through a bridge developed by Matter Labs. However, this feature has not been completed at the time of the audit start date, making it subject to future audits.

Overall, we deem the codebase as robust, but lacking certain input restrictions (SOP-2, SOP-3). A flow has been identified that could lead to user funds being lost (SOP-1). The test suite thoroughly tests the codebase. While documentation was sparse and specifications of the protocol architecture were missing, we were able to get clarification via a code walkthrough. The client has also been very responsive with our additional questions after the walkthrough.

Update Fix-Review All issues have been either fixed or acknowledged. A few tests have been added that further improve the existing test suite.

ID	DESCRIPTION	SEVERITY	STATUS
SOP-1	**Potential Lost Funds when Depositing ETH and Converting to sDAI Principal**	• High	Fixed
SOP-2	**Missing Enforcement of Precision Increase on `pointsPerBlock` Could Cause Nullified Rewards**	• Medium	Fixed
SOP-3	**Missing Input Validation Could Cause Reverts**	• Medium	Fixed
SOP-4	**Adding And Point Adjusting Of Pools Can Cause Loss of Points**	• Medium	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
SOP-5	Multiple CEI-Pattern Violations	● Low	Fixed
SOP-6	Owner Can Reactivate Farming Once Ended	● Low	Fixed
SOP-7	weETH and wstETH Share Initial Point Allocation Even Though Underlying Worth in USD Differs	● Informational	Fixed
SOP-8	Application Monitoring Can Be Improved by Emitting More Events	● Informational	Fixed
SOP-9	Fee-on-Transfer and Rebasing Tokens Not Supported for Lp Token	● Informational	Acknowledged
SOP-10	Privileged Roles and Ownership	● Informational	Acknowledged
SOP-11	Missing Storage Gaps	● Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/sophon-org/farming-contracts(a4d4c0b1b7df74c35606ba4c2d3034883c170da2) Files:

- contracts/farm/SophonFarming.sol
- contracts/farm/SophonFarmingState.sol

Files Excluded

All other files.

Furthermore, the client specifically instructed us that all aspects around the bridge design present in the current version of the contracts are to be completed at a future time and hence out of scope of the audit. This includes the `bridgePool()` and `revertFailedBridge()` functions in the `SophonFarming.sol` file.

Findings

SOP-1
Potential Lost Funds when Depositing ETH and Converting to sDAI Principal

• High ⓘ Fixed

Update
Marked as "Fixed" by the client. The relevant flow is now reverting.
Addressed in: `04fdedacc1e7037a77ac58a934680b3bc32e9cfa`.

File(s) affected: `SophonFarming.sol`

Description: With the `depositEth()` and `depositWeth()` functions, users can exchange native ETH/wETH into one of three predefined asset pools: wstETH, weETH or sDAI. The function is callable with `PredefinedPool.sDAI`, but does not perform a conversion from (w)ETH to DAI, as it is assumed by the subsequent call to `_depositPredefinedAsset()`. As DAI also has 18 decimals, users could accidentally swap 1 (w)ETH into 1 DAI of sDAI, so roughly one USD, assuming that the contract has an existing balance of 1 DAI.

Recommendation: Either add a conversion from ETH to DAI in the `depositEth()` and `depositWeth()` function if `_predefinedPool == PredefinedPool.sDAI` or revert if that selection is taken.

SOP-2
Missing Enforcement of Precision Increase on `pointsPerBlock` Could Cause Nullified Rewards

• Medium ⓘ Fixed

Update
Marked as "Fixed" by the client. A lower bound of `1e18` has been added for the `pointsPerBlock` variable as recommended.
Addressed in: `f0b82fd83a5d85eb8dc7ba2bbce2d49fefb326a4`.

File(s) affected: `SophonFarming.sol`

Description: For the calculations of `accPointsPerShare` in the `updatePool()` function to not possibly nullify rewards due to precision loss, it needs to be the case that `pointReward >> lpSupply`:

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

`pointReward` is a percentage of the multiplication of `blockMultiplier` and `pointsPerBlock`. Both values are expected to have a precision increase of `1e18`. However, that precision increase is only enforced for the `blockMultiplier` variable, not for `pointsPerBlock`.

If `pointsPerBlock` were to not be precision increased, `lpSupply` (assuming 18 decimals in the corresponding LPToken) might be greater than `pointReward`, causing the rewards accumulated since the last pool update to get nullified.

Recommendation: Enforce the `1e18` precision in the `pointsPerBlock` variable in the same manner as it is done for the `boosterMultiplier` variable. If `pointsPerBlock = 0` is a valid case, then enforce a value of either zero or a value greater than `1e18`.

SOP-3 Missing Input Validation Could Cause Reverts

• Medium ⓘ Fixed

Update
Marked as "Fixed" by the client. An upper limit of `10001e18` for `pointsPerBlock` and `10e18` for `boosterMultiplier` has been added.

Addressed in: `639e80bc9ca94c085a41be8001336a094e58b82b`.

File(s) affected: `SophonFarming.sol`

Description: Technically, the `boosterMultiplier` and `pointsPerBlock` are unbounded `uint256` and can therefore be set to arbitrary values. They can be configured in a way to cause integer overflows and deny block withdrawals. They may break pools indefinitely, if pools have rewards settled with a number close to `uint256.max`, causing subsequent accounting to overflow in the process.

Furthermore, most of the address-inputs from owner-controlled functions are unchecked for the zero address.

Recommendation: Consider adding reasonable input validation.

SOP-4
Adding And Point Adjusting Of Pools Can Cause Loss of Points

● Medium ⓘ Fixed

Update
Marked as "Fixed" by the client. All pools are now always updated as part of calls to `add()` and `set()`.
Addressed in: `6d117397d57a39a865cee0621cabf9154fb6fd71`.

File(s) affected: `SophonFarming.sol`

Description: Pools accumulate rewards based on their share of `totalAllocPoints`. For example, in 424, we can see that the point rewards are based on the number of passed blocks, multiplied by the point emissions per block and multiplied with the relative share of the pool's `allocPoints`. If adjustments to the allocations are made, it should be preceded by an update of all pools, so all already accumulated rewards get calculated and accounted with the old and for the `totalAllocPoints` value valid for the period since the last reward settlement. However, both in the `SophonFarming.set()` and `SophonFarming.add()` function, the mass-update of pools is only optional.

If a new pool is added via `SophonFarming.add()` without the option to mass-update the other pools, the already existing, rewards accumulating pools are not updated prior. This could be problematic if there are a set of pools that have not been updated in a while, so that do not have had their already accrued rewards locked in.

This can lead to incorrect accounting, because all the up to this moment accrued rewards will be shared to the corresponding pools based on their share of the `totalAllocPoint`, essentially their share of the rewards. However, the newly added pool will increase the `totalAllocPoint`, effectively diluting each existing pool, so once the "old" pools will solidify their rewards, they will get their reward share based on this increased `totalAllocPoint`, resulting in a reduced share because of the potentially significantly later added pool. This results in essentially this newly added or allocation-updated pool getting an inaccessible share of the rewards of each pool since their last update. However, this share is inaccessible to any party and will therefore remain forever locked in the contract.

In the `set()` function, the outcome is more severe than in the `add()` function. Without a prior pool update in the `set()` function with deactivated mass-update, already accumulated, but not yet accounted rewards of a pool would fail to be accounted at all, since a call to `updatePool()` is missing that would increase `rewardSettled`. However, it still increases the `pool.lastRewardBlock` to the current timestamp. This leads to the pool essentially nullifying all the rewards accumulated since the last pool update, as with the timestamp update and the missing `rewardSettled` update, all rewards up to the current block are then considered settled.

Recommendation: Make a call to `massUpdatePools()` mandatory in the `add()` and `set()` function.

SOP-5 Multiple CEI-Pattern Violations

● Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `036403fd88e82d5acdc1d91230fbc1663cbc757c`.

File(s) affected: `SophonFarming.sol`

Description: Functions including external calls should always follow the Check, Effects, Interactions-pattern to avoid misconfigurations with possible reentrancies, especially since the contract does not employ reentrancy guards.

- In the `initialize()` function, `initialized = true` should be set after the checks and before the external calls.
- In the `withdraw()` function, the lp tokens should be transferred after the update to `user.rewardDebt`
- In the `bridgePool()` function, `isBridged = true` should be set after the checks and before the external calls.

It should be noted that the deposit flows also violate the CEI-pattern with the preliminary external calls. This could technically lead to incorrect accounting and stuck tokens in the contract.

Recommendation: Follow the CEI pattern in the listed functions.

SOP-6 Owner Can Reactivate Farming Once Ended

● Low ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `7a21a5086b8e99611d7225c86209e8d150c9c64a`.

File(s) affected: `SophonFarming.sol`

Description: If set, once the specified `endBlock` is reached, reward emission is supposed to stop. However, due to an incorrect ordering of checks, the `SophonFarming.setEndBlock()` function only reverts if farming has ended if the newly proposed `_endBlock` timestamp is unequal to zero. Therefore, the owner can theoretically extend the farming, possibly after several pools have already been bridged. This would also grant rewards retroactively since the previous `endBlock` timestamp for all pools that are not yet migrated.

Theoretically this can put the contract in a situation where a pool is bridged, yet still has active farming, so can still be deposited into and is still accruing rewards.

Recommendation: In the `setEndBlock()` function, perform the `isFarmingEnded()` before the branching.

SOP-7
weETH and wstETH Share Initial Point Allocation Even Though Underlying Worth in USD Differs

● Informational ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `bb67b53b0f5940b3f5bc9b23caca9ff03a0f65f2`.

File(s) affected: `SophonFarming.sol`

Description: In the `SophonFarming.initialize()` function, the weETH and wstETH-pools share the same weighing of `ethAllocPoint_` for the allocation point assignment. However, the stETH-pool should most likely carry a slightly higher point allocation, since a unit of stETH is worth more than eETH, since it has been accruing staking rewards for a vastly longer time. The current price difference is about 400USD. Therefore, it would be more profitable for users to deposit weETH into the protocol than wstETH under the initial allocation point allocation.

Recommendation: Consider having separate variables for the assignment of the initial allocation points of the pools and give wstETH a slightly higher share in the allocation points.

SOP-8
Application Monitoring Can Be Improved by Emitting More Events

● Informational ⓘ Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `941496374bc3f9770f2cdfc136f140d02982979d`.

File(s) affected: `SophonFarming.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:

- `revertFailedBridge()` should emit a `RevertFailedBridge` event
- `updatePool()` should emit a `PoolUpdated` event

Recommendation: Consider emitting the events.

SOP-9
Fee-on-Transfer and Rebasing Tokens Not Supported for Lp Token

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
No plans to create pools for Fee-on-Transfer and Rebasing Tokens.
```

File(s) affected: `SophonFarming.sol`

Description: The `SophonFarming` contract would incorrectly account some deposit balances in case some rebasing/elastic or fee-on-transfer tokens would be a registered LPToken. It is worthy to note that the contract does interact with rebasing tokens, namely stETH, eETH and DAI, but only in an intermediary stage before swapping them to wrapped, accruing versions and depositing those into the designated pool.

Not enabling such tokens for LPTokens would only be a very minor restriction that will not effect UX.

Recommendation: We mainly want to raise awareness for this fact. Avoid adding pools with rebasing ERC20 tokens or with tokens including a on-transfer-fee.

SOP-10 Privileged Roles and Ownership

● Informational Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Noted
```

File(s) affected: `SophonFarming.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. This is the case in the given contracts:

- The owner has control over the initial allocation points for predefined pool via the one-time callable `initialize()` function and can also later on adjust these values for all pools via `set()`. It is worth noting that rewards can not be retroactively reduced in any way, as rewards are settled to the current timestamp before updates.
- The owner can update the bridge contract to use via `setBridge()` and the contract to which a certain pool will be migrated to via `setL2FarmForPool()`. The inputs are unchecked and can be set to any address at any time.
- The owner can update the start timestamp of the farming via `setStartBlock()`. The start timestamp has to be in the future, so rewards can not be retroactively awarded.
- The owner can update the end timestamp and an optional withdrawal period of the farming via `setEndBlock()`. The end timestamp can only be set to a future timestamp or be removed entirely. It can however be shorter than the previously set end timestamp. It can therefore be repeatedly extended. If a withdraw period is set, users are not able to withdraw after that timestamp hasn been reached. An existing withdraw period can be shortened prior to the end of the farming.
- The owner can call `setPointsPerBlock()`, with which the passed-time-multiplier for future point distribution can be adjusted.
- The owner can call `setBoosterMultiplier()`, with which the multiplier for future point distribution for boosted deposit unit can be adjusted
- The owner can call `revertFailedBridge()`, intended to possibly retry failed pool-bridge attempts. This would enable the contract to re-execute a call to the bridge with the recorded pool token balance, even if they original call might have already been successful.
- The owner can call `withdrawProceeds()` to make the LP token sacrificed by users for boosted rewards redeemable to the owner.

Furthermore, owner-controlled variables can theoretically be configured in a way to cause integer overflows and deny block withdrawals, possibly indefinitely, if pools has rewards settled with some astronomical number

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

SOP-11 Missing Storage Gaps

● Informational Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Care will always be taken during upgrades to ensure there are no storage collisions. By convention,
state variables will never be added directly to SophonFarming.sol, but rather to the end of the stack
in SophonFarmingState.sol. We will not modify the types of existing variables in the stack.
```

File(s) affected: `SophonFarming.sol`

Description: The `SophonFarming` contract is designed to be upgradable. Upgradable contracts usually have reserved space to allow future versions to add new state variables to the contract without shifting down storage in the inheritance chain. As `SophonStateFarming` is inherited by `SophonFarming`, adding a storage gap is necessary for preventing storage collisions in future updates.

Recommendation: When writing new upgradable contracts, consider including a storage gap. See OpenZeppelin's documentation for more details.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Adherence to Best Practices

1. Fixed The comments in the `SophonFarmingState` contract are not in line with the contract's behavior, as they are still based on the MasterChef-fork. For example, the described calculation of pending rewards and the process of user deposits is inaccurate.
2. Acknowledged Generally, we discourage setting of max-approvals for external contracts, as in case of their compromise, the tokens held by this contract will potentially become drainable.
3. Acknowledged ETH can be directly converted to wstETH by transferring the ETH directly to the wstETH-contract instead of first converting it to stETH (link).
4. Acknowledged `string` and `byteX` parameters can be of type `calldata` instead of `memory` if they remain immutable in the function's execution. For example, in `SophonFarming.add()`, the `description` parameter can be set to `calldata`.
5. Acknowledged Struct declarations can be of type `memory` if they are only being read from, e.g. `pool` and `user` in `SophonFarming._pendingPoints()`.
6. Fixed In for loops, `array.length` can be cached before to save gas. This is done in some functions, but not consistently in the code.
7. Fixed Since Solidity version 0.8.22, the optimization regarding unchecked counter increases in for loop is no longer necessary.
8. Fixed In `SophonFarming.withdrawProceeds()`, the `pool` variable can be of type `memory`.
9. Mitigated Functions that are not called within the contract and are marked as `public` can be marked as `external` for reduced deployment costs, e.g. the `SophonFarming.set()` function.
10. Fixed The `getPoolInfo()` function can simply return the `poolInfo` variable instead of recreating the array in memory.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `dd7...cef ./farm/SophonFarmingState.sol`
- `764...a62 ./farm/SophonFarming.sol`
- `f32...3a6 ./farm/interfaces/IeETHLiquidityPool.sol`
- `242...17c ./farm/interfaces/IstETH.sol`
- `1ee...195 ./farm/interfaces/IsDAI.sol`
- `425...933 ./farm/interfaces/IWeth.sol`
- `38f...85d ./farm/interfaces/IwstETH.sol`
- `8cc...496 ./farm/interfaces/IweETH.sol`

Tests

- `a70...c29 ./test/SophonFarming.t.sol`
- `9be...c8f ./test/utils/SophonFarmingHarness.sol`

Automated Analysis

N/A

Test Suite Results

Test output was obtained with executing `forge test`. The test suite is of high quality, containing 86 tests, including fuzz tests.

Update Fix-Review: The test suite has been adjusted with the contract changes and four additional tests have been added.

```
Ran 88 tests for test/SophonFarming.t.sol:SophonFarmingTest
[PASS] testFuzz_Add(uint256) (runs: 256, μ: 1001811, ~: 1001811)
[PASS] testFuzz_DepositDai_BoostedDeposit(uint256,uint256) (runs: 256, μ: 406293, ~: 416486)
[PASS] testFuzz_DepositDai_BoostedWithdraw(uint256,uint256,uint256) (runs: 256, μ: 574750, ~: 574958)
[PASS] testFuzz_DepositDai_MaxWithdraw(uint256) (runs: 256, μ: 494625, ~: 494626)
[PASS] testFuzz_DepositDai_NotBoostedDeposit(uint256) (runs: 256, μ: 364753, ~: 364755)
[PASS] testFuzz_DepositDai_NotBoostedWithdraw(uint256,uint256) (runs: 256, μ: 504778, ~: 509812)
[PASS] testFuzz_DepositDai_RevertWhen_NotBoostedWithdraw_InvalidWithdraw(uint256) (runs: 256, μ: 379150,
~: 379152)
[PASS] testFuzz_DepositDai_RevertWhen_WithdrawTooHigh(uint256) (runs: 256, μ: 406520, ~: 406520)
[PASS] testFuzz_DepositEEth_Boosted(uint256,uint256) (runs: 256, μ: 427836, ~: 440640)
[PASS] testFuzz_DepositEEth_NotBoosted(uint256) (runs: 256, μ: 391302, ~: 391304)
[PASS] testFuzz_DepositEEth_RevertWhen_BoostTooHigh(uint256) (runs: 256, μ: 296677, ~: 296679)
[PASS] testFuzz_DepositEEth_RevertWhen_FarmingIsEnded(uint256,uint256) (runs: 256, μ: 314845, ~: 314846)
[PASS] testFuzz_DepositEEth_RevertWhen_InvalidDeposit() (gas: 261592)
[PASS] testFuzz_DepositEthToWeETH_NotBoosted(uint256) (runs: 256, μ: 263353, ~: 263353)
[PASS] testFuzz_DepositEth_Boosted(uint256,uint256) (runs: 256, μ: 259389, ~: 272293)
[PASS] testFuzz_DepositEth_MultipleNotBoosted(uint256,uint256) (runs: 256, μ: 260080, ~: 247080)
[PASS] testFuzz_DepositEth_NotBoosted(uint256) (runs: 256, μ: 210602, ~: 210602)
[PASS] testFuzz_DepositEth_RevertWhen_BoostTooHigh(uint256) (runs: 256, μ: 93201, ~: 93201)
[PASS] testFuzz_DepositEth_RevertWhen_FarmingIsEnded(uint256,uint256) (runs: 256, μ: 115976, ~: 115976)
[PASS] testFuzz_DepositEth_RevertWhen_InvalidDeposit(uint256,uint256) (runs: 256, μ: 66631, ~: 66741)
[PASS] testFuzz_DepositEth_RevertWhen_NoEthSent(uint256,uint256) (runs: 256, μ: 12838, ~: 12838)
[PASS] testFuzz_DepositStEth_Boosted(uint256,uint256) (runs: 256, μ: 400956, ~: 413139)
[PASS] testFuzz_DepositStEth_NotBoosted(uint256) (runs: 256, μ: 363790, ~: 363792)
[PASS] testFuzz_DepositStEth_RevertWhen_BoostTooHigh(uint256) (runs: 256, μ: 269224, ~: 269227)
[PASS] testFuzz_DepositStEth_RevertWhen_FarmingIsEnded(uint256,uint256) (runs: 256, μ: 287502, ~: 287503)
[PASS] testFuzz_DepositStEth_RevertWhen_InvalidDeposit() (gas: 247926)
[PASS] testFuzz_DepositWethToWeETH_NotBoosted(uint256) (runs: 256, μ: 297921, ~: 297921)
[PASS] testFuzz_DepositWeth_Boosted(uint256,uint256) (runs: 256, μ: 296367, ~: 306799)
[PASS] testFuzz_DepositWeth_NotBoosted(uint256) (runs: 256, μ: 255528, ~: 255530)
[PASS] testFuzz_DepositWeth_RevertWhen_BoostTooHigh(uint256) (runs: 256, μ: 160918, ~: 160920)
[PASS] testFuzz_DepositWeth_RevertWhen_FarmingIsEnded(uint256,uint256) (runs: 256, μ: 179016, ~: 179018)
[PASS] testFuzz_DepositWeth_RevertWhen_InvalidDeposit(uint256,uint256) (runs: 256, μ: 123572, ~: 123634)
[PASS] testFuzz_DepositWstEth_Boosted(uint256,uint256) (runs: 256, μ: 365209, ~: 377392)
[PASS] testFuzz_DepositWstEth_NotBoosted(uint256) (runs: 256, μ: 328081, ~: 328083)
[PASS] testFuzz_DepositWstEth_RevertWhen_BoostTooHigh(uint256) (runs: 256, μ: 235969, ~: 235971)
[PASS] testFuzz_DepositWstEth_RevertWhen_FarmingIsEnded(uint256,uint256) (runs: 256, μ: 254976, ~:
254977)
[PASS] testFuzz_DepositWstEth_RevertWhen_InvalidDeposit() (gas: 203274)
[PASS] testFuzz_IncreaseBoost(uint256,uint256) (runs: 256, μ: 386051, ~: 386050)
[PASS] testFuzz_IncreaseBoost_RevertWhen_BoostTooHigh(uint256) (runs: 256, μ: 330033, ~: 330032)
[PASS] testFuzz_SetBoosterMultiplier(uint256) (runs: 256, μ: 31272, ~: 31283)
[PASS] testFuzz_SetEndBlock(uint256) (runs: 256, μ: 31109, ~: 31109)
[PASS] testFuzz_SetFunction(uint256) (runs: 256, μ: 121317, ~: 121317)
[PASS] testFuzz_SetPointsPerBlock(uint256) (runs: 256, μ: 34899, ~: 34943)
[PASS] testFuzz_SetStartBlock(uint256) (runs: 256, μ: 21927, ~: 21927)
[PASS] testFuzz_WithdrawProceeds(uint256,uint256) (runs: 256, μ: 383359, ~: 393925)
[PASS] test_Add_RevertWhen_FarmingIsEnded() (gas: 801424)
[PASS] test_Add_RevertWhen_PoolExists() (gas: 43574)
[PASS] test_Add_RevertWhen_ZeroAddress() (gas: 39284)
[PASS] test_BecomeImplementation() (gas: 3376591)
[PASS] test_BecomeImplementation_RevertWhen_OwnableUnauthorizedAccount() (gas: 3393260)
[PASS] test_BecomeImplementation_RevertWhen_Unauthorized() (gas: 3392997)
[PASS] test_BridgePool() (gas: 475025)
[PASS] test_BridgePool_RevertWhen_BridgeInvalid() (gas: 54333)
```

```
[PASS] test_BridgePool_RevertWhen_Unauthorized() (gas: 16561)
[PASS] test_ConstructorParameters() (gas: 24445)
[PASS] test_GetBlockMultiplier() (gas: 34358)
[PASS] test_GetBlockNumber() (gas: 8958)
[PASS] test_GetMaxAdditionalBoost(uint256) (runs: 256, μ: 326103, ~: 326104)
[PASS] test_GetPoolInfo() (gas: 106702)
[PASS] test_IncreaseBoost_RevertWhen_BoostIsZero() (gas: 13322)
[PASS] test_IncreaseBoost_RevertWhen_FarmingIsEnded() (gas: 39158)
[PASS] test_Initialize() (gas: 134276)
[PASS] test_Initialize_RevertWhen_AlreadyInitialized() (gas: 15553)
[PASS] test_Initialize_RevertWhen_InvalidBooster() (gas: 3413181)
[PASS] test_Initialize_RevertWhen_InvalidPointsPerBlock() (gas: 3750686)
[PASS] test_Initialize_RevertWhen_InvalidStartBlock() (gas: 3772916)
[PASS] test_IsFarmingEnded() (gas: 39524)
[PASS] test_IsWithdrawPeriodEnded() (gas: 36566)
[PASS] test_MassUpdatePools() (gas: 112209)
[PASS] test_MultipleDeposits(uint256) (runs: 256, μ: 1452912, ~: 1452914)
[PASS] test_PoolLength() (gas: 10501)
[PASS] test_Proxy() (gas: 3564110)
[PASS] test_ReplaceImplementation() (gas: 3389302)
[PASS] test_RevertFailedBridge() (gas: 459328)
[PASS] test_SetBoosterMultiplier_RevertWhen_InvalidBooster() (gas: 36621)
[PASS] test_SetBoosterMultiplier_RevertWhen_IsFarmingEnded() (gas: 36905)
[PASS] test_SetBridge() (gas: 115697)
[PASS] test_SetBridge_RevertWhen_ZeroAddress() (gas: 13388)
[PASS] test_SetEndBlock_RevertWhen_FarmingIsEnded() (gas: 36945)
[PASS] test_SetEndBlock_RevertWhen_InvalidEndBlock() (gas: 24384)
[PASS] test_SetL2FarmForPool() (gas: 106774)
[PASS] test_SetL2FarmForPool_RevertWhen_ZeroAddress() (gas: 16543)
[PASS] test_SetPointsPerBlock_RevertWhen_InvalidPointsPerBlock() (gas: 15471)
[PASS] test_SetPointsPerBlock_RevertWhen_IsFarmingEnded() (gas: 36860)
[PASS] test_SetStartBlock_RevertWhen_FarmingIsStarted() (gas: 18079)
[PASS] test_SetStartBlock_RevertWhen_InvalidStartBlock() (gas: 39901)
[PASS] test_Set_RevertWhen_FarmingIsEnded() (gas: 40069)
[PASS] test_UpdatePool() (gas: 106787)
Suite result: ok. 88 passed; 0 failed; 0 skipped; finished in 2.31s (16.13s CPU time)

Ran 1 test suite in 2.46s (2.31s CPU time): 88 tests passed, 0 failed, 0 skipped (88 total tests)
```

Code Coverage

Test output was obtained with executing `forge coverage`. The coverage metrics of the contracts are nearly perfect.

Update Fix-Review: Metrics continue to look very good.

File	% Lines	% Statements	% Branches	% Funcs
contracts/farm/SophonFarming.sol	97.95% (**286**/292)	98.38% (**365**/371)	94.00% (**94**/100)	100.00% (**42**/42)
Total	97.95% (**286**/292)	98.38% (**365**/371)	94.00% (**94**/100)	100.00% (**42**/42)

Changelog

- 2024-05-22 - Initial report
- 2024-06-04 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

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

Quantstamp