Quantstamp Security Assessment Certificate


October 29th 2022 — Quantstamp Verified

# Merit Circle Audit

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

## Executive Summary

| | |
|---|---|
| Type | Liquidity Mining |
| Auditors | Faycal Lalidji, Senior Security Engineer<br>Roman Rohleder, Research Engineer<br>Mostafa Yassin, Security Engineer |
| Timeline | 2022-10-17 through 2022-10-24 |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review |
| Specification | MIP-19 Upgrading to staking V2 |
| Documentation Quality | Medium |
| Test Quality | Low |

Source Code

| Repository | Commit |
|---|---|
| Merit-Circle/merit-liquidity-mining | ce5feaa initial audit |

| | |
|---|---|
| Total Issues | **10** (5 Resolved) |
| High Risk Issues | **2** (2 Resolved) |
| Medium Risk Issues | **2** (1 Resolved) |
| Low Risk Issues | 0 (0 Resolved) |
| Informational Risk Issues | **6** (2 Resolved) |
| Undetermined Risk Issues | 0 (0 Resolved) |


| | |
|---|---|
| High Risk | The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users. |
| Medium Risk | The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact. |
| Low Risk | 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. |

| | |
|---|---|
| Unresolved | Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it. |
| 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). |
| Fixed | Adjusted program implementation, requirements or constraints to eliminate the risk. |
| Mitigated | Implemented actions to minimize the impact or likelihood of the risk. |

## Summary of Findings

**Initial Audit:**
We have found multiple issues ranging from High to informational severity that need to be fixed before deployment. We highly recommend implementing sufficient testing to possibly verify all the project specifications.
**Fix Review Update:**
All raised issues have been either fixed or acknowledged by the Merit Circle team except for QSP-7 which was mitigated.

| ID | Description | Severity | Status |
|---|---|---|---|
| QSP-1 | Inconsistent Accounting for `unit` when Increasing/decreasing the Curve Through `TimeLockPool.setCurvePoint()` | High | Fixed |
| QSP-2 | Unchecked Curve Values Can Lead Users Transactions to Fail | High | Fixed |
| QSP-3 | Unlimited Approval in `BasePool.__BasePool_init()` | Medium | Acknowledged |
| QSP-4 | Potential Re-Entrancy/checks-Effects-Interactions Pattern Violations | Medium | Fixed |
| QSP-5 | The Use of `SafeERC20.safeApprove()` | Informational | Acknowledged |
| QSP-6 | Privileged Roles and Ownership | Informational | Acknowledged |
| QSP-7 | Missing Input Validation | Informational | Mitigated |
| QSP-8 | Application Monitoring Can Be Improved by Emitting More Events | Informational | Acknowledged |
| QSP-9 | Upgrade-Able Contracts Do Not Contain Gaps | Informational | Fixed |
| QSP-10 | `View.fetchOldData()` Might Return Incorrect Values | Informational | Acknowledged |

## Quantstamp Audit Breakdown

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

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features 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**

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. 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:
   i. 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.
   ii. 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, clarify, 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.

**Toolset**

The notes below outline the setup and steps performed in the process of this audit.

**Setup**

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

# Findings

## QSP-1 Inconsistent Accounting for `unit` when Increasing/decreasing the Curve Through `TimeLockPool.setCurvePoint()`

**Severity:** *High Risk*

**Status:** Fixed

**File(s) affected:** `TimeLockPool.sol`

**Description:** When increasing or decreasing the `TimeLockPool.curve[]` array using `setCurve()`, the `unit` variable gets updated correspondingly (`unit = maxLockDuration / (curve.length - 1);`). However, when increasing or decreasing the curve through the function `setCurvePoint()`, similar updates are missing to `unit`, leading to an inconsistent state of that variable, relative to the new length of the curve.

**Recommendation:** We recommend adding a corresponding update to the `unit` variable in the cases where the length of the `curve[]` array changes in the function `setCurvePoint()`.

**Update:** Fixed in commit "c0d4a00".

## QSP-2 Unchecked Curve Values Can Lead Users Transactions to Fail

**Severity:** *High Risk*

**Status:** Fixed

**File(s) affected:** `TimeLockPool.sol`

**Description:** The protocol relies on a curve along with linear interpolation to give the rewards for a certain point in time that is on the curve. For that to happen, the curve values need to be **strictly** increasing. However, there is no implemented checks in `TimeLockPool.__TimeLockPool_init()`, `TimeLockPool.setCurve()` and `TimeLockPool.setCurvePoint()`.
In such case, the line below will throw due to underflow and prevent users from staking, since:

```
return 1e18 + curve[n] + (_lockDuration - n * unit) * (curve[n + 1] - curve[n]) / unit;
```

**Recommendation:** In all functions that input a new curve or a curve point value, a new point `curve[n]` has to be checked to be greater than `curve[n-1]` and lower than `curve[n+1]`.

**Update:** Fixed in commit "8de2048".

## QSP-3 Unlimited Approval in `BasePool.__BasePool_init()`

**Severity:** *Medium Risk*

**Status:** Acknowledged

**File(s) affected:** `BasePool.sol`

**Description:** The contract approves the address of the escrow pool to transfer tokens on its behalf without setting a limit on how many tokens may be transferred (`IERC20(_rewardToken).safeApprove(_escrowPool, type(uint256).max);`). If the approved address becomes hacked or is intentionally malicious, it may transfer out all of the approved tokens.

**Recommendation:** We recommend removing unlimited approvals and approving only the amount that needs to be transferred in a given transaction.

**Update:** This issue was acknowledged by the Merit Circle team: "This is done not to repeatedly approve the escrow pool while claiming rewards. The risk of being hacked for both contracts is the same as they are both governed by the multi-sig".

## QSP-4 Potential Re-Entrancy/checks-Effects-Interactions Pattern Violations

**Severity:** *Medium Risk*

**Status:** Fixed

**File(s) affected:** `BasePool.sol`, `TimeLockPool.sol`

**Related Issue(s):** SWC-107

**Description:** As a best practice and to prevent unwanted external contract side effects, it is advised to adhere to the "Checks-Effects-Interactions"-pattern, even in the presence of reentrancy guards.
The following instances were observed where said pattern was violated:

- `BasePool.distributeRewards()`: Performs external contract calls (`rewardToken.safeTransferFrom()`), before modifying state variables in following sub-calls.
- `TimeLockPool.deposit()`: Performs external contract calls (`depositToken.safeTransferFrom()`), before modifying state variables in following sub-calls.
- `TimeLockPool.increaseLock()`: Performs external contract calls (`depositToken.safeTransferFrom()`), before modifying state variables in following sub-calls.

**Recommendation:** Consider re-structuring the code by executing all the external calls at the end of the function logic, such that it no longer violates the "Checks-Effects-Interactions"-pattern and/or add reentrancy guards at corresponding calling locations.

**Update:** Fixed in commit "60615e4".

## QSP-5 The Use of `SafeERC20.safeApprove()`

**Severity:** *Informational*

**Status:** Acknowledged

**File(s) affected:** `BasePool.sol`

**Description:** The code relies on function `SafeERC20.safeApprove()`. The function shall not be used instead of `approve()` since, 1) it can still be front-run, 2) uses additional gas, and 3) does not work with tokens that are not ERC-20 compliant. For more information see this issue.

**Recommendation:** Use `approve()`or `safeIncreaseAllowance()` and `safeDecreaseAllowance()` if the recommendation in "Unlimited Approval in `BasePool.__BasePool_init()`" are followed.

**Update:** This issue was acknowledged by the Merit Circle team: "As a consequence of QSP-3 being acknowledged and not fixed nor mitigated, this finding will not be fixed nor mitigated".

## QSP-6 Privileged Roles and Ownership

Severity: *Informational*

**Status:** Acknowledged

**File(s) affected:** `BasePool.sol`, `TimeLockPool.sol`

**Description:** Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.

The `BasePool.sol` contract contains the following privileged roles:

- `DEFAULT_ADMIN_ROLE`, as initialized during the `constructor()` execution to `msg.sender`:
  - Assign the `GOV_ROLE` to arbitrary addresses by calling `grantRole()`.

The `TimeLockPool.sol` contract contains the following privileged roles (besides the one listed for `BasePool.sol` from which this contract inherits):

- `GOV_ROLE`, as set through `grantRole()` by `DEFAULT_ADMIN_ROLE`:
  - Renounce the role (**and thereby prevent any future calls to the followingly listed functions!**) by calling `renounceRole()`.
  - Set entirely new curve parameters by calling `setCurve()`.
  - Change the value of an existing curve point (adding a new point or removing the last point) by calling `setCurvePoint()`.

**Recommendation:** Clarify the impact of these privileged actions on the end-users via publicly facing documentation.

**Update:** This issue was acknowledged by the Merit Circle team: "These high privilege admin powers will be governed by the multisig and later be migrated to the onchain Merit Circle DAO once fully operational".

## QSP-7 Missing Input Validation

Severity: *Informational*

**Status:** Mitigated

**File(s) affected:** `BasePool.sol`, `TimeLockPool.sol`

**Description:** It is important to validate inputs to avoid human error, even if they only come from trusted addresses. The following functions do not have a proper validation of input parameters:

1. [UNRESEOLVED ] `BasePool.__BasePool_init()` does not check that parameter `_escrowDuration` is different from zero (or otherwise bound).
2. [UNRESEOLVED ] `TimeLockPool.__TimeLockPool_init()` does not check that parameter `_maxBonus` is different from zero (or otherwise bound).
3. [FIXED ] `TimeLockPool.withdraw()` does not check that parameter `_receiver` is different from `address(0)`.

**Recommendation:** Consider adding to required input checks.

**Update:** Fixed in commit "1e521cb".

## QSP-8 Application Monitoring Can Be Improved by Emitting More Events

Severity: *Informational*

**Status:** Acknowledged

**File(s) affected:** `AbstractRewards.sol`, `TimeLockPool.sol`

**Description:** To validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transition 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:

1. `AbstractRewards._distributeRewards()` does not emit an event reflecting changes made to the state variable `pointsPerShare`.
2. `AbstractRewards._prepareCollect()` does not emit an event reflecting changes made to the state variable `withdrawnRewards[]`.
3. `AbstractRewards._correctPointsForTransfer()` does not emit an event reflecting changes made to the state variable `pointsCorrection[]`.
4. `AbstractRewards._correctPoints()` does not emit an event reflecting changes made to the state variable `pointsCorrection[]`.
5. `TimeLockPool.deposit()` does not emit an event reflecting changes made to the state variable `depositsOf[].start` (`block.timestamp`).

**Recommendation:** Consider emitting the events.

**Update:** This issue was acknowledged by the Merit Circle team: "Choosing not to emit events for these specific variables are part of the intended design of the contracts".

## QSP-9 Upgrade-Able Contracts Do Not Contain Gaps

Severity: *Informational*

**Status:** Fixed

**File(s) affected:** `BasePool.sol`, `AbstractRewards.sol`, `BoringBatchable.sol`

**Description:** Gaps are used to allow for adding future state variables when an upgrade is required. Base contracts with no gaps will cause storage collisions if a new state variable is added.

**Recommendation:** If adding new storage variables might be needed for the base contracts, consider adding `uint256` gap arrays to reserve storage slots.

**Update:** Fixed in "f1ae899".

### QSP-10 `View.fetchOldData()` Might Return Incorrect Values

*Severity: Informational*

**Status:** Acknowledged

**File(s) affected:** `View.sol`

**Description:** `View.fetchOldData()`execute `getMulitplier()` to fetch the reward multiplier values. If the old pool curve or other info can be updated after deprecating the previous pool version, the returned value of `getMulitplier()` can be incorrect.

**Recommendation:** We recommend documenting this behavior or disabling any possible admin function that can modify the previous pool states.

**Update:** This issue was acknowledged by the Merit Circle team: "This function is aimed to fetch data from taking V1 pools that can not be updated, making getMultiplier compatible with this method".

## Automated Analyses

### Slither

Slither did not raise any significant findings.

## Code Documentation

1. The following typographical errors have been noted:
    1. `AbstractRewards.sol#L68` and `IAbstractRewards.sol#L21`: `accumulativeFundsOf` -> `cumulativeRewardsOf`.
    2. `TimeLockPool.sol#L78-79`: Unintelligible sentence.
    3. `TimeLockPool.sol#L111`: `withdrawl` -> `withdrawal`.
    4. `TimeLockPool.sol#L113`: `increased` -> `withdrawn from`.
    5. `TimeLockPool.sol#L141`: `increase` -> `increased` and `results is a` -> `results in a`.
    6. `TimeLockPool.sol#L143`: `correspondant` -> `corresponding`.
    7. `TimeLockPool.sol#L146`: `meassured` -> `measured`.
    8. `TimeLockPool.sol#L171`: `if` -> `is`.
    9. `TimeLockPool.sol#L212`: `acording` -> `according`.
    10. `TimeLockPool.sol#L212`: `to the deposit to end` -> `from the deposit until its end`.
    11. `TimeLockPool.sol#L240`: `de` -> `the`.
    12. `TimeLockPool.sol#L275`: `new` -> `new one` and `For` -> `By`.
    13. `TimeLockPool.sol#L317`: `greated` -> `greater`.
2. Missing or incorrect NatSpec comments:
    1. Duplicate NatSpec comments for public functions in `AbstractRewards.sol` and `IAbstractRewards.sol`. Consider re-using the comments from the interface, by only using the keyword `/// @inheritdoc IAbstractRewards`.
    2. `TimeLockPool.sol#L83`: `uint256` -> `address`.
    3. `TimeLockPool.sol#L114`: `uint256` -> `address`.
    4. `TimeLockPool.withdraw()`: Incorrect NatSpec comment for parameter `_receiver` (It is the receiver of the withdrawn funds, not the `owner of the lock`).

## Adherence to Best Practices

1. To improve readability and lower the risk of introducing errors when making code changes, it is advised not to use magic constants throughout code, but instead, declare them once (as constant and commented) and use these constant variables instead. The following instances should therefore be changed accordingly:
    1. `BasePool.sol#L63` and `L102` and `TimeLockPool.sol#L96`, `L167`, `L215`, `L242` and `L245`: `1e18`.

2. For usability and maintainability interfaces should declare all public/external functions of the implementation contract. In this regard consider the following cases:
    1. External function `BasePool.claimRewards()` not declared in `IBasePool.sol`.
    2. External function `TimeLockPool.withdraw()` not declared in `ITimeLockPool.sol`.
    3. External function `TimeLockPool.extendLock()` not declared in `ITimeLockPool.sol`.
    4. External function `TimeLockPool.increaseLock()` not declared in `ITimeLockPool.sol`.
    5. Public function `TimeLockPool.getMultiplier()` not declared in `ITimeLockPool.sol`.
    6. Public function `TimeLockPool.getTotalDeposit()` not declared in `ITimeLockPool.sol`.
    7. Public function `TimeLockPool.getDepositsOf()` not declared in `ITimeLockPool.sol`.
    8. Public function `TimeLockPool.getDepositsOfLength()` not declared in `ITimeLockPool.sol`.
    9. Public function `TimeLockPool.getDepositsOfLength()` not declared in `ITimeLockPool.sol`.

3. Before rolling out code in production, any pending `TODO` items in code should be resolved in order not to deploy potentially unfinished code. In this regard the following

`TODO` items remain in code and should be resolved:

1. `TimeLockPool.sol#L240`: `// TODO check if this is needed`.

4. `AbstractRewards.POINTS_MULTIPLIER` is unnecessarily high: using a high decimals value as a multiplier is useful. However, if the listed tokens decimals are high enough, transactions can revert due to overflow.

## Test Results

### Test Suite Results

```
Network Info
============
> HardhatEVM: v2.6.2
> network:    hardhat

Generating typings for: 0 artifacts in dir: typechain for target: ethers-v5
Successfully generated 87 typings!

  BasePool
    distributeRewards
       ✓ Should fail when there are no shares
       ✓ Should fail when tokens are not approved (260ms)
       ✓ Should work (199ms)
    claimRewards
       ✓ First claim single holder (285ms)
       ✓ Claim multiple holders (508ms)
       ✓ Multiple claims, distribution and holders (758ms)
       ✓ Zero escrow (283ms)
       ✓ Full escrow (585ms)

TimeLockNonTransferablePool
   ✓ transfer
   ✓ transferFrom

TimeLockPool
  deposit
     ✓ Depositing with no lock should lock it for 10 minutes to prevent flashloans (79ms)
     ✓ Deposit with no lock (101ms)
     ✓ Trying to lock for longer than max duration should lock for max duration (95ms)
     ✓ Multiple deposits (163ms)
     ✓ Should fail when transfer fails (73ms)
  withdraw
     ✓ Withdraw to zero address should fail
     ✓ Withdraw before expiry should fail
     ✓ Should work (256ms)
  extendLock
     ✓ Extending with zero duration should fail
     ✓ Extending when deposit has already expired should fail
     ✓ Extending should emit event with the correct arguments (214ms)
     ✓ Extending should change start and extend end time in the struct (221ms)
     ✓ Extending in between end and start should change start and extend end time in the struct (216ms)
     ✓ Extending should mint correct amount of tokens and change shareAmount in the struct (227ms)
     ✓ Extending in between end and start should mint correct amount of tokens and change shareAmount in the struct (547ms)
  increaseLock
     ✓ Increasing with zero amount should fail
     ✓ Increasing when deposit has already expired should fail
     ✓ Increasing should emit event with the correct arguments (65ms)
     ✓ Increasing should mint correct amount of tokens and change shareAmount in the struct (83ms)
     ✓ Increasing in between start and end of deposit should do it correctly (87ms)
  getMultiplier
     ✓ Left curve point should be relative to minimum time
     ✓ Right curve point should be relative to max time
     ✓ An intermediate time value should create a proportional intermediate multiplier
  setCurve and setCurvePoint
     ✓ Replacing a curve with a non gov role account should fail
     ✓ Replacing a curve should emit an event (47ms)
     ✓ Replacing with a same length curve should do it correctly (80ms)
     ✓ Replacing with a shorter curve should do it correctly (173ms)
     ✓ Replacing with a longer curve should do it correctly (227ms)
     ✓ Replacing with a curve that is not monotonic increasing should fail (41ms)
     ✓ Replacing a point with a non gov role account should fail
     ✓ Replacing a point should emit an event
     ✓ Replacing a point should do it correctly (39ms)
     ✓ Adding a point should do it correctly (45ms)
     ✓ Removing a point should do it correctly (108ms)
     ✓ Removing a point from a curve with length two should revert (174ms)
     ✓ Adding a point so that curve is not monotonic increasing should fail
  Curve changes
     ✓ Curve should multiply correctly
     ✓ Change curve and multiply correctly (84ms)
     ✓ Change curve by extending it (189ms)
     ✓ Change curve by reducing it (191ms)
  Curve changes: withdawing/increasing/extending
     ✓ Withdrawing after curve change should work correctly (283ms)
     ✓ Extending lock with a new curve should do it correctly (243ms)
     ✓ Extending lock with a significant smaller new curve should burn tokens (239ms)
  Batchable
     ✓ User should make multiple deposits on the same transaction (1142ms)
     ✓ User should be able to withdraw and deposit in the same transaction (911ms)
     ✓ User should be able to increase and extend lock in the same transaction (544ms)
     ✓ Gov should be able to set curve in batches (394ms)
     ✓ Should revert after a failed call (161ms)
     ✓ Should not revert after a failed call (222ms)
  View
     ✓ Should retrieve correct information from a user from one pool (163ms)

TimeLockPool
  upgradeable
    proxyAdmin
       ✓ Should set correctly the proxy admin
       ✓ Should set correctly the implementation
       ✓ Should have governance as owner
       ✓ Should have governance as owner
    upgrade
       ✓ Should set another implementation correctly with it's functions (88ms)
       ✓ Should preserve the deposits in the same slot after upgrade (1096ms)
       ✓ Should preserve storage when extending (634ms)
       ✓ Should preserve storage when changing curve (663ms)
       ✓ Should find a slot that changed (741ms)

69 passing (18s)
```

## Code Coverage

**Initial Summary**
Quantstamp usually recommends developers increase the branch coverage to 90% and above before a project goes live to avoid hidden functional bugs that might not be easy to spot during the development phase. For branch code coverage, the current targeted files by the audit achieve a lower score that should be enhanced before deployment.
**Fix Review Summary** The code branch coverage after the fix review is still under the recommended 90%.

| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
|---|---|---|---|---|---|
| contracts/ | 92.31 | 79.17 | 94.44 | 86.26 | |
| ProxyAdmin.sol | 100 | 100 | 100 | 100 | |
| TimeLockNonTransferablePool.sol | 100 | 100 | 100 | 100 | |
| TimeLockPool.sol | 100 | 79.17 | 100 | 91.89 | … 284,327,347 |
| TransparentUpgradeableProxy.sol | 100 | 100 | 100 | 100 | |
| View.sol | 50 | 100 | 50 | 50 | … 71,72,73,81 |
| contracts/base/ | 89.66 | 77.27 | 88.89 | 85.71 | |
| AbstractRewards.sol | 84.21 | 66.67 | 87.5 | 85 | 102,103,104 |
| BasePool.sol | 93.75 | 83.33 | 87.5 | 88.89 | 68,71,95,96 |
| BoringBatchable.sol | 85.71 | 75 | 100 | 71.43 | 19,23 |
| contracts/interfaces/ | 100 | 100 | 100 | 100 | |
| IAbstractRewards.sol | 100 | 100 | 100 | 100 | |
| IBasePool.sol | 100 | 100 | 100 | 100 | |
| ITimeLockPool.sol | 100 | 100 | 100 | 100 | |
| **All files** | **91.43** | **78.57** | **91.67** | **86.08** | |

## 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.

#### Contracts

```
670ca45c96e2ae170ce467fae0ff7ba8623eaaaf42db31226a5898f70c20878a ./contracts/TransparentUpgradeableProxy.sol
6aafc067bc9569af82bf02ab64fd6f267144deba5496da340be5e6c8ba221ba6 ./contracts/TimeLockPool.sol
6b393396929569dda876ca04cfd6eef8eab7674c77c4e91d3c76b9b35f15d34a ./contracts/View.sol
25e6917a73330d14181ebd2e5f905731c05b6e9665da6b3e7ad8d0fcd4a0db5d ./contracts/TimeLockNonTransferablePool.sol
a56fcaba6433e187ff4c946e44bdfb32234e24903f1962dadd2acda8a3a1842e ./contracts/ProxyAdmin.sol
ab8a41e10edc145f9777345cc75add4b90e52a2b34257323531cfe9685b26426 ./contracts/base/BasePool.sol
1304aa8a94e1760972c001bbe754f3ea6b7ee7a3463f3a624cfd191aad40552a ./contracts/base/AbstractRewards.sol
ef5187e40e505e56f19ade75a9e6154ac997fd658fc736b050c80e2ed4c95564 ./contracts/base/BoringBatchable.sol
e4d54710f7d465f7b264f10c571373c078dce11e2c677bf334220fcd97f68d0b ./contracts/interfaces/IAbstractRewards.sol
7490e734dccc420440f73fda9faa95500e8a56c64ed38535788cd9a78bde4440 ./contracts/interfaces/IBasePool.sol
6ed743f409d47a1fd57d6cfd82c63a9d4780e18e92718eef0822c19eb47492ae ./contracts/interfaces/ITimeLockPool.sol
34b79e96ba58a220965725b4f4c3b3350f06ef6330242b07e5ec80101cc20106 ./contracts/test/TestFaucetToken.sol
dc79250ac1a086a86e43daa8d7e5a0833f01013ea94298b933c1f6165b56872a ./contracts/test/TestToken.sol
018a7b881aa8b0c8505cc6721e955feecc8f93cdf1078029077b5df72e5afe99 ./contracts/test/TestTimeLockPool.sol
37ef3593c461a0ca1e3b641aadd80e84ddbb33dadcc2f8b7b7e223efab09b974 ./contracts/test/TimeLockNonTransferablePoolV2.sol
2683fcd57de99d6045991f998c01f223d476a891f4df30e04f96c4f1e4705229 ./contracts/test/TestBasePool.sol
```

#### Tests

```
893abac6a12d4e42f37fd4a7e15047517b58c838dd2169149503d9e19ee3e945 ./test/BasePool.ts
70cf6c24e1d20d43a6331bb295102df593bcdcf9227124df34a0772818419fad ./test/Upgradeable.ts
7f9831ea658abeeb04717946c65f7ed1d0c0622560b4cadb319cf983e8a52736 ./test/TimeLockPool.ts
827ea0f9a515e01e2a62633505eff2df5b7bbe39ba3d41a3ef0bab92d76b5fb0 ./test/TimeLockNonTransferablePool.ts
```

# Changelog

- 2022-10-20 - Initial report
- 2022-10-29 - Fix review

## About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors 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 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

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, Aave, 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