Quantstamp Security Assessment Certificate

November 4th 2021 — Quantstamp Verified

Merit Circle

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

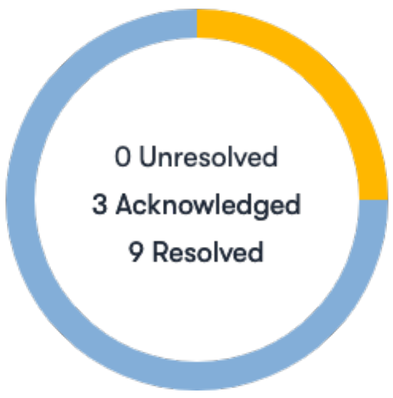
Executive Summary

Type	DeFi Protocol
Auditors	Jan Gorzny, Blockchain Researcher Roman Rohleder, Research Engineer
Timeline	2021-10-28 through 2021-10-28
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Undetermined
Test Quality	Undetermined

Source Code

Repository	Commit
merit-liquidity-mining	f558820

Total Issues	**12** (9 Resolved)
High Risk Issues	**4** (4 Resolved)
Medium Risk Issues	**1** (0 Resolved)
Low Risk Issues	**3** (2 Resolved)
Informational Risk Issues	**1** (0 Resolved)
Undetermined Risk Issues	**3** (3 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).
Resolved	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

Quantstamp has reviewed the Merit Liquidity Mining repository. Quantstamp found several issues which have all been addressed. Some issues were unavoidable due to the design of the system and were acknowledged, while others were fixed. The code is accompanied by tests, but Quantstamp was unable to compute the coverage provided by those tests (due to the project's use of Hardhat).

ID	Description	Severity	Status
QSP-1	Write to Arbitrary Storage Location	High	Fixed
QSP-2	Use of Insecure Casting Operations	High	Fixed
QSP-3	Unchecked Return Values	High	Fixed
QSP-4	Flash Loan Vulnerability	High	Fixed
QSP-5	Maximum Approve	Medium	Acknowledged
QSP-6	Unlocked Pragma	Low	Fixed
QSP-7	Privileged Roles and Ownership	Low	Acknowledged
QSP-8	Missing Input Validation	Low	Fixed
QSP-9	Events Not Emitted on State Change	Informational	Acknowledged
QSP-10	`View.fetchData()` Always Calling `getMultiplier(0)`	Undetermined	Fixed
QSP-11	Gas Costs for Processing Arrays Could be Prohibitive	Undetermined	Fixed
QSP-12	Ignored Failed Transaction	Undetermined	Fixed

Quantstamp Audit Breakdown

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

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 v0.6.6

Steps taken to run the tools:

Installed the Slither tool: `pip install slither-analyzer` Run Slither from the project directory: `slither .`

Findings

QSP-1 Write to Arbitrary Storage Location

Severity: *High Risk*

Status: Fixed

File(s) affected: `TimeLockPool.sol`, `LiquidityMiningManager.sol`

Related Issue(s): SWC-124

Description: In `LiquidityMiningManager.removePool()` parameter `_poolId` is not checked to be within the bounds of `pools.length`, allowing the caller to write the contents of `pools[pools.length - 1]` to an arbitrary storage location, by carefully crafting `_poolId`, when executing L75 of `LiquidityMiningManager.sol`. Similarly, but in this case performable by anyone and not just an address having the `GOV_ROLE` role, in `TimeLockPool.withdraw()` `_depositId` is not checked to be within the bounds of `depositsOf[_msgSender()].length`, allowing the caller to write the contents of `depositsOf[_msgSender()][depositsOf[_msgSender()].length - 1]` to an arbitrary storage location, by carefully crafting `_depositId`, when executing L71 of `TimeLockPool.sol`. See https://github.com/Arachnid/uscc/tree/master/submissions-2017/doughoyte for reference.

Recommendation: Add a check for `_poolId` to be smaller than `pools.length` and `_depositId` smaller than `depositsOf[_msgSender()].length` in `LiquidityMiningManager.removePool()` and `TimeLockPool.withdraw()` respectively.

Update: The relevant checks have been added.

QSP-2 Use of Insecure Casting Operations

Severity: *High Risk*

Status: Fixed

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

Related Issue(s): SWC-101

Description: In `BasePool._mint()`, `BasePool._burn()` and `TimeLockPool.withdraw()` the insecure primitive casting operations `int256()` and `uint256()` are used. For sufficiently large positive or negative values these casts may wrap around, without leading to a revert and therefore lead to unexpected behaviour.
In function `cumulativeRewardsOf` of `AbstractRewards.sol`, a `UInt256` is cast to an `Int256` variable, which may not be safe (Line 71).

Recommendation: Replace the use of these insecure cast operations with for example their secure counterparts of OpenZeppelins SafeCast library.

Update: The use of these insecure cast operations has been replaced by those of the `SafeCast` library.

QSP-3 Unchecked Return Values

Severity: *High Risk*

Status: Fixed

File(s) affected: `TokenSaver.sol`, `BasePool.sol`, `LiquidityMiningManager.sol`

Related Issue(s): SWC-104

Description: The following function calls make a call to a function that returns a value, which however is not checked:

- `LiquidityMiningManager.sol`,
  - `addPool()` ignores the return value of `approve()` (Line 62).
  - `distributeRewards()` ignores the return value of `transferFrom()` (Line 116).
  - `distributeRewards()` ignores the return value of `transfer()` (Line 129), `call()` (Line 122).
- `TokenSaver.sol`:
  - `saveToken()` not checking return value of `IERC20(_token).transfer()`.
- `BasePool.sol`:
  - `claimRewards()` not checking return value of `rewardToken.transfer(_receiver, nonEscrowedRewardAmount);`.
  - the `constructor` ignores the return value of `approve()` (Line 42).

Recommendation: Add checks at above mentioned call sites, i.e. by wrapping the calls within corresponding `require(...)` statements or use the safe counterparts (i.e. `safeTransfer()` instead of `transfer()`), where applicable.

Update: The calls have been replaced by their `safe` variants.

QSP-4 Flash Loan Vulnerability

Severity: *High Risk*

Status: Fixed

Description: Oversimplifying, a *flash loan* is a way to atomically borrow some tokens, performs some actions using them, and repays the initial loan at the end of the transaction. A flash loan *attack* is a way to use flash loans to extract an unfair amount of value from a system.
In this system, since rewards are a function of a user's deposits, a quick injection of funds may entitle users to a larger share of rewards. Immediately withdrawing these funds may mean that those rewards are taken from other participants and were not earned honestly.

Recommendation: Ensure that funds are locked for some minimum, non-trivial time period, or protect against flash loans using another method.

Update: This has been addressed in the commit listed in this report (it was discovered prior to report compilation and resolved before an initial report was sent).

QSP-5 Maximum Approve

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `LiquidityMiningManager.sol`

Description: Line 62 calls `approve(_poolContract, type(uint256).max);` which means unlimited funds can be moved if something goes wrong.

Recommendation: Design this out, or make sure users are aware of this requirement.

Update: This has been acknowledged; from the team: "Intended behavior. Doing the approval on every reward distribution would add significant gas costs. Contracts added as pools are trusted. Additionally the `LiquidityMiningManager` does not hold significant funds at any time as those are always send back after every distribution. Also the distribution of rewards can only be called by a trusted address."

QSP-6 Unlocked Pragma

Severity: *Low Risk*

Status: Fixed

File(s) affected: `IAbstractRewards.sol`, `TimeLockPool.sol`, `IBasePool.sol`, `View.sol`, `LiquidityMiningManager.sol`, `ITimeLock.sol`, `AbstractRewards.sol`,, `BasePool.sol`, `TokenSaver.sol`, `TimeLockNonTransferablePool.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

Update: The pragma has been locked.

QSP-7 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

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

Description: Certain contracts have special roles, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.
The owner of the `TimeLockPool.sol` or `TimeLockNonTransferablePool.sol` contracts may perform the following privileged actions:

1. Give or revoke the role of `TOKEN_SAVER_ROLE` to any arbitrary address.
2. Call `saveToken()`, thereby transferring an arbitrary amount of an arbitrary token from the current contract to an arbitrary address.
3. Renounce ownership, by calling `renounceOwnership()`, thereby preventing the change of the currently set `TOKEN_SAVER_ROLE` role.
4. Transfer ownership (the role of `DEFAULT_ADMIN_ROLE`) to an arbitrary address.

The owner of the `LiquidityMiningManager.sol` contract may perform the following privileged actions:

1. Give or revoke the role of `TOKEN_SAVER_ROLE` to any arbitrary address.
2. Call `saveToken()`, thereby transferring an arbitrary amount of an arbitrary token of the `LiquidityMiningManager.sol` contract to an arbitrary address.
3. Give or revoke the role of `GOV_ROLE` to any arbitrary address.
4. Add or remove pools, change pool weights or `rewardPerSecond`, by calling `addPool()`, `removePool()`, `adjustWeight()` and `setRewardPerSecond()` respectively.
5. Give or revoke the role of `REWARD_DISTRIBUTOR_ROLE` to any arbitrary address.
6. Distribute rewards by calling distributeRewards()` .
7. Renounce ownership, by calling `renounceOwnership(),` thereby preventing the change of the currently set `TOKEN_SAVER_ROLE`, `GOV_ROLE` and `REWARD_DISTRIBUTOR_ROLE` roles.
8. Transfer ownership (the role of `DEFAULT_ADMIN_ROLE`) to an arbitrary address.

Note: As functions `addPool()`, `removePool()` and `adjustWeight()` call `distributeRewards()`, which is only callable by a reward distributor role holding account, it entails that the `GOV_ROLE` holding account will also hold the role of `REWARD_DISTRIBUTOR_ROLE`.

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

Update: This issue has been acknowledged. From the team: "Privileged roles will be documented in public facing documentation".

QSP-8 Missing Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `View.sol`, `LiquidityMiningManager.sol`, `AbstractRewards.sol`, `TimeLockPool.sol`, `BasePool.sol`

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

1. `BasePool.constructor()` does not check that `_depositToken`is not the zero address, thus neither does the constructors of `TimeLockPool.sol` and `TimeLockNonTransferablePool.sol`.
2. `View.constructor()` does not check that parameters `_liquidityMiningManager` and `_escrowPool` are different from `address(0)`.
3. `LiquidityMiningManager.constructor()` does not check that parameters `_reward` and `_rewardSource` are different from `address(0)`.
4. `LiquidityMiningManager.addPool()` does not check that parameter `_poolContract` is different from `address(0)` or parameter `_weight` is different from zero.
5. `LiquidityMiningManager.adjustWeight()` does not check that parameter `_poolId` is smaller than `LiquidityMiningManager.pools.length` or that parameter `_newWeight` is non-zero.

6. `AbstractRewards.constructor` does not check that parameters `getSharesOf_` and `getTotalShares_` are different from `address(0)`.
7. `AbstractRewards._prepareCollect()` does not check that parameter `_account` is different from `address(0)`.
8. `AbstractRewards._correctPointsForTransfer()` does not check that parameters `_from` and `_to` are different from `address(0)` or `_shares` is non-zero.
9. `AbstractRewards._correctPoints()` does not check that parameter `_account` is different from `address(0)` or _shares is non-zero.
10. `TimeLockPool.constructor()` does not check that parameter `_maxLockDuration` is greater than `MIN_LOCK_DURATION` (10min).
11. `TimeLockPool.deposit()` does not check that parameters `_amount` and `_duration` are non-zero or that parameter `_receiver` is different from `address(0)`.
12. `TimeLockPool.withdraw()` does not check that parameter `_depositId` is within `depositsOf[_msgSender()].length`, or that parameter `_receiver` is different from `address(0)`.

Recommendation: Check that the values are not the obviously incorrect values, or clarify that those values are acceptable.

Update: Updates:

1. Fixed.
2. Acknowledged: "`View.sol` contract is only used as an easy way to fetch data. When the addresses in the constructor are incorrect it will simply not work. No need to fix and increase gas const on deploy."
3. Fixed.
4. Fixed/acknowledged: "Pool contract now checked when adding pool. Being able to add a pool at zero weight is intended."
5. Fixed/acknowledged: "Existence of pool is now checked, Being able to set a pools weight to 0 is intended."
6. False positive (fixed) / acknowledged: "`getSharesOf_` and `getTotalShares_` are not of the type address. They are functions. The only contract directly inheriting from `AbtractRewards` which properly passes `balanceOf` and `totalSupply` to the constructor.
7. False positive (fixed) / acknowledged: "`_prepareCollect` is only called in `BasePool.claimRewards` and uses `_msgSender()` as the account parameter. Which can never be `address(0)`. Additionally passing `address(0)` would not be an issue. Doing a check for `address(0)` would needlessly waste gas."
8. Acknowledged: "Future inheriting contracts should be able to use `address(0)` as they see fit. example burning tokens to send them to `address(0)`. In the current situation from can never be `address(0)` as the OpenZeppelin token implementation prevents transfers from/to those addresses."
9. Acknowledged: "`address(0)` is like any other address. Correcting points for that address would not cause any issues. Also `_mint` and `_burn` in the OpenZeppelin token implementation already catch the zero address.
10. Fixed.
11. Fixed: "`_amount` now checked. `_duration` is mutated to be above 0 when doing `duration = duration.max(MIN_LOCK_DURATION);`, OpenZeppelin ERC20 `_mint` prevents mint to account `address(0)`."
12. Fixed: "Deposit length now being checked. If users explicitly want to burn their rewards that's fine. But reward burning will be prevented by the MC token as it doesn't allow transfers to `address(0)`."

QSP-9 Events Not Emitted on State Change

Severity: *Informational*

Status: Acknowledged

File(s) affected: `AbstractRewards.sol`

Description: An event should always be emitted when a state change is performed in order to facilitate smart contract monitoring by other systems which want to integrate with the smart contract. This is not the case for the functions:

1. `AbstractRewards._correctPointsForTransfer()` does not emit any event upon a successful change of the state variables `pointsCorrection[_from]` and `pointsCorrection[_to]`.
2. `AbstractRewards._correctPoints()` does not emit any event upon a successful change of the state variable `pointsCorrection[_account]`.

Recommendation: Emit an event in the aforementioned functions.

Update: This issue has been acknowledged. From the team: "Point correction is only used internally inside the contract to track rewards. Any function that does it already emits relevant events. No need for additional gas usage to emit event."

QSP-10 `View.fetchData()` Always Calling `getMultiplier(0)`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `View.sol`

Description: In `View.fetchData()` the `multiplier` field of the Deposit structures in L80 and L107 are set with `multiplier: poolContract.getMultiplier(deposit.end - deposit.end)` and `multiplier: escrowPool.getMultiplier(deposit.end - deposit.end)`. As `deposit.end - deposit.end` is used it will always result in zero, always returning the same multiplier `1e18`.

Recommendation: To clarify if this is intended behaviour or if rather `deposit.end - deposit.start` should have been used.

Update: This has been resolved by using `deposit.end - deposit.start` as suggested.

QSP-11 Gas Costs for Processing Arrays Could be Prohibitive

Severity: *Undetermined*

Status: Fixed

File(s) affected: `View.sol`, `TimeLockPool.sol`, `LiquidityMiningManager.sol`

Related Issue(s): SWC-128

Description: In `View.fetchData()` and `LiquidityMiningManager.distributeRewards()` a for-loop is used to iterate over the pools returned by

`liquidityMiningManager.getPools()`. Similarly, `TimeLockPool.depositsOf`, which is user-controllable through deposits and iterated over in `TimeLockPool.getTotalDeposit()`. For sufficiently large arrays, processing over these arrays could run out of gas.

Recommendation: Consider adding a check on the current `pool` array length, i.e. in `LiquidityMiningManager.addPool()` and `TimeLockPool.deposit()`, to prevent arrays/number of pools and deposits larger than a set maximum being processed. This maximum can be determined by performing gas analysis.

Update: This has been fixed and acknowledged. From the team: "Limited pool count in to 10 which would in current conditions would allow more than 1M gas to be consumed per pool. `getTotal` deposits is only used from non contracts as a view method which allow much higher gas usage generally. Additionally it is not mission critical but more of an utility."

QSP-12 Ignored Failed Transaction

Severity: *Undetermined*

Status: Fixed

File(s) affected: `LiquidityMiningManager.sol`

Description: There is a comment `ignore tx failing` on line 121. The effects of ignoring a failed transaction is not clear.

Exploit Scenario:

Recommendation: Handle a failed transaction or clarify the behaviour when this happens.

Update: The issue has been resolved by describing the rationale of ignoring the failed transaction. Added comment: "Ignore tx failing to prevent a single pool from halting reward distribution".

Automated Analyses

Slither

Slither reported many issues, most of which are false positives or have been placed into other parts of this report. The reentrancy it reported is replicated below (as it does not appear elsewhere), and only concerns the `distributeRewards` function, which is only callable by an account with the `REWARD_DISTRIBUTOR_ROLE` role. Since this role is assumed to be trusted (see QSP-7), these are false positives.

```
Reentrancy in LiquidityMiningManager.addPool(address,uint256) (contracts/LiquidityMiningManager.sol#48-65):
    External calls:
    - distributeRewards() (contracts/LiquidityMiningManager.sol#49)
        - reward.transferFrom(rewardSource,address(this),totalRewardAmount) (contracts/LiquidityMiningManager.sol#116)
        - address(pool.poolContract).call(abi.encodeWithSelector(pool.poolContract.distributeRewards.selector,poolRewardAmount)) (contracts/LiquidityMiningManager.sol#122)
        - reward.transfer(rewardSource,leftOverReward) (contracts/LiquidityMiningManager.sol#129)
    State variables written after the call(s):
    - pools.push(Pool(IBasePool(_poolContract),_weight)) (contracts/LiquidityMiningManager.sol#52-55)
    - totalWeight += _weight (contracts/LiquidityMiningManager.sol#59)

Reentrancy in LiquidityMiningManager.adjustWeight(uint256,uint256) (contracts/LiquidityMiningManager.sol#82-92):
    External calls:
    - distributeRewards() (contracts/LiquidityMiningManager.sol#83)
        - reward.transferFrom(rewardSource,address(this),totalRewardAmount) (contracts/LiquidityMiningManager.sol#116)
        - address(pool.poolContract).call(abi.encodeWithSelector(pool.poolContract.distributeRewards.selector,poolRewardAmount)) (contracts/LiquidityMiningManager.sol#122)
        - reward.transfer(rewardSource,leftOverReward) (contracts/LiquidityMiningManager.sol#129)
    State variables written after the call(s):
    - pool.weight = _newWeight (contracts/LiquidityMiningManager.sol#89)
    - totalWeight -= pool.weight (contracts/LiquidityMiningManager.sol#86)
    - totalWeight += _newWeight (contracts/LiquidityMiningManager.sol#87)

Reentrancy in LiquidityMiningManager.removePool(uint256) (contracts/LiquidityMiningManager.sol#67-80):
    External calls:
    - distributeRewards() (contracts/LiquidityMiningManager.sol#68)
        - reward.transferFrom(rewardSource,address(this),totalRewardAmount) (contracts/LiquidityMiningManager.sol#116)
        - address(pool.poolContract).call(abi.encodeWithSelector(pool.poolContract.distributeRewards.selector,poolRewardAmount)) (contracts/LiquidityMiningManager.sol#122)
        - reward.transfer(rewardSource,leftOverReward) (contracts/LiquidityMiningManager.sol#129)
    State variables written after the call(s):
    - pools[_poolId] = pools[pools.length - 1] (contracts/LiquidityMiningManager.sol#75)
    - pools.pop() (contracts/LiquidityMiningManager.sol#76)
    - totalWeight -= pools[_poolId].weight (contracts/LiquidityMiningManager.sol#72)
```

Code Documentation

1. The NatSpec comment for `AbstractRewards._distributeRewards()` mentions it reverts if if the total supply is 0, however it reverts if shares is zero. **Update:** fixed.
2. The NatSpec comment for `AbstractRewards._distributeRewards()` mentions it emits `FundsDistributed`, however it emits `RewardsDistributed`. **Update:** fixed.
3. The revert message for `AbstractRewards._distributeRewards()` has a typo. Instead of share "suppy" it should be share "supply". **Update:** fixed.

Adherence to Best Practices

1. For improved readability it is recommended to have a maximum line length of 79 or 99. Therefore L28, L33 and L122 of `LiquidityMiningManager.sol`, L17 of `TimeLockNonTransferablePool.sol`, L36, L41, L42, L68 and L71 of `TimeLockPool.sol`, L8, L11, L15, L16, L17, L65, L71 and L121 of `AbstractRewards.sol`, L23 and L80 of `BasePool.sol`, L11 and L22 of `TokenSaver.sol`, which exceed these limits, should be shortened accordingly.
2. To prevent confusion it is recommended to avoid re-using the same/similar names for different variables, functions or structures. Contract `View.sol` defines structures `Deposit` and `Pool`, which however are different from the structures `Deposit` in `TimeLockPool.sol` and `Pool` from `LiquidityMiningManager.sol` and should therefore be renamed.
3. The state variable `TimeLockPool.MIN_LOCK_DURATION` has the immutable modifier and is immediately initialized at declaration. Consider replacing the immutable modifier with `constant`, as it is more in line with the immediate initialization to a constant value.
4. It is discouraged to use uncommented magic constants in code, as their role may be non-apparent and using such constants without having them globally defined may lead to inconsistencies in future changes. In L34 and L68 of `BasePool.sol` and L58, L68 and L83 of `TimeLockPool.sol` `1e18` is used. Consider declaring it a constant and comment it, i.e., in `BasePool.sol` and replace the corresponding uses.

Test Results

Test Suite Results

```
BasePool
  distributeRewards
     ✓ Should fail when there are no shares
     ✓ Should fail when tokens are not approved (285ms)
     ✓ Should work (266ms)
  claimRewards
```

```
      ✓ First claim single holder (263ms)
      ✓ Claim multiple holders (440ms)
      ✓ Multiple claims, distribution and holders (588ms)
      ✓ Zero escrow (234ms)
      ✓ Full escrow (472ms)

  LiquidityMiningManager
    Adding pools
      ✓ Adding a single pool (59ms)
      ✓ Adding multiple pools (94ms)
      ✓ Adding a pool twice should fail (46ms)
      ✓ Adding a pool from a non gov address should fail
    Removing pools
      ✓ Removing last pool in list (94ms)
      ✓ Removing a pool in the beginning of the list (82ms)
      ✓ Removing all pools (329ms)
      ✓ Removing a pool from a non gov address should fail
    Distributing rewards
      ✓ Distributing rewards from an address which does not have the REWARD_DISTRIBUTOR_ROLE
      ✓ Distributing zero rewards
      ✓ Should return any excess rewards (601ms)
      ✓ Should work (531ms)
    Adjusting weight
      ✓ Adjust weight up
      ✓ Adjust weight down (68ms)
      ✓ Should fail from non gov address
    Setting reward per second
      ✓ Should work
      ✓ Should fail from non gov address

  TimeLockNonTransferablePool
     ✓ transfer
     ✓ transferFrom

  TimeLockPool
    deposit
      ✓ Depositing with no lock should lock it for 10 minutes to prevent flashloans (171ms)
      ✓ Deposit with no lock (197ms)
      ✓ Trying to lock for longer than max duration should lock for max duration (187ms)
      ✓ Multiple deposits (352ms)
      ✓ Should fail when transfer fails (45ms)
    withdraw
      ✓ Withdraw before expiry should fail
      ✓ Should work (159ms)

  TokenSaver
    saveToken
      ✓ Should fail when called fron non whitelised address
      ✓ Should work (112ms)


  36 passing (10s)
```

Code Coverage

Quantstamp was unable to compute code coverage for the tests, due to the project's use of hardhat.

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

```
32e20dc9c7834eb1abcb11483bc16b9467e6329d27fa4ea0f310bfdc854052c2 ./contracts/LiquidityMiningManager.sol
65d66fa08c13c10901a59a4c3c19d9c1075396715491e8b4f48d659bf5dd77c5 ./contracts/TimeLockNonTransferablePool.sol
cc36e4786ebc928814354f1cc6d43bac72b48c4d9102678daaf55f9f8dc263c7 ./contracts/TimeLockPool.sol
780b387f9c2639481484774bd6addf11a89359936fe17e06b237cc18077f02e7 ./contracts/View.sol
44129dc0fb197cdc38891d05da506588492794f89f6de83902a96f3b6d621281 ./contracts/test/TestBasePool.sol
34b79e96ba58a220965725b4f4c3b3350f06ef6330242b07e5ec80101cc20106 ./contracts/test/TestFaucetToken.sol
dc79250ac1a086a86e43daa8d7e5a0833f01013ea94298b933c1f6165b56872a ./contracts/test/TestToken.sol
e4d54710f7d465f7b264f10c571373c078dce11e2c677bf334220fcd97f68d0b ./contracts/interfaces/IAbstractRewards.sol
7490e734dccc420440f73fda9faa95500e8a56c64ed38535788cd9a78bde4440 ./contracts/interfaces/IBasePool.sol
6ed743f409d47a1fd57d6cfd82c63a9d4780e18e92718eef0822c19eb47492ae ./contracts/interfaces/ITimeLockPool.sol
bbc65efeb36fe3b5674fb7774270323aa9c1ad9a6d1517a206bdcfed648b3eea ./contracts/base/AbstractRewards.sol
2d466b469fd6079fdca2305b8716320d460c40e2060a6dc081f76bd0b223eab7 ./contracts/base/BasePool.sol
4df1f949bfcddf7305dfba1ef6021842105ebd1c793a5c440217af60f69743b2 ./contracts/base/TokenSaver.sol
```

Tests

```
9f0de342cf41c8b92dbb62b0ac1f38b2c21d0f49772c6a642a0203812e50429d ./test/BasePool.ts
f6f9a85335a9e82dabac2fbf4f3701ba117dc1eb10403e2a5fc4ad8278ea5372 ./test/LiquidityMiningManager.ts
fcf0835789d668fb0c08cef4bec813750973efea1752f3cb0bd210f3f08a9919 ./test/TimeLockNonTransferablePool.ts
0d8fdac6533eb46dc25a37a4f1e74a3d78645ed1a638de370a84db3ba5b6fd2b ./test/TimeLockPool.ts
a837563927a505dbd90499489111b95898caefdcf292df9865d7202e1602e65c ./test/TokenSaver.ts
```

Changelog

- 2021-10-28 - Initial report [`bc1ef21`]
- 2021-10-31 - Revised report [`f558820`]

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.