Quantstamp Security Assessment Certificate

August 3rd 2022 — Quantstamp Verified

Badger - Rewards Manager

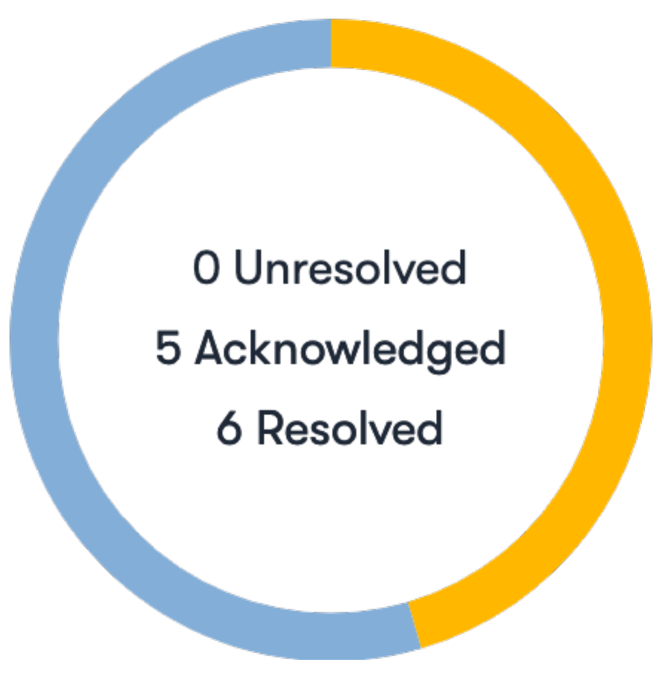
This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Rewards Manager
Auditors	Zeeshan Meghji, Auditing Engineer Ibrahim Abouzied, Auditing Engineer Poming Lee, Senior Research Engineer
Timeline	2022-07-05 through 2022-07-15
EVM	Gray Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Notion Page
Documentation Quality	High
Test Quality	High
Source Code	

Repository	Commit
badger-onchain-rewards (initial)	f534011
badger-onchain-rewards (re-audit)	bcfd392

Total Issues	**11**	(6 Resolved)
High Risk Issues	**2**	(2 Resolved)
Medium Risk Issues	**1**	(1 Resolved)
Low Risk Issues	**2**	(1 Resolved)
Informational Risk Issues	**5**	(2 Resolved)
Undetermined Risk Issues	**1**	(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

The RewardsManager contract integrates with vaults to manage their users' rewards. Rewards can be added to the RewardsManager contract by anyone. Those rewards can be claimed directly from the RewardsManager contract by the vault's depositors. Rewards accumulate per epoch, each of which has a duration of one week.

During the audit, we found 11 vulnerabilities ranging in severity from high to informational. The most critical issues (QSP-1, QSP-5) involved vaults claiming significantly more than their allocated rewards. Another significant finding (QSP-2) was that users' shares could be burned when claiming rewards with certain parameters. We also noted that some validation checks were missing (QSP-3). The contract was well-documented through the Notion page and through code comments. We recommended documenting the code further using the NatSpec standard.

Update: Following the re-audit, all reported issues have been either fixed or acknowledged. We found the Badger team to be responsive and cooperative regarding the recommendations made by the auditors.

ID	Description	Severity	Status
QSP-1	One Vault Can Steal All Rewards	High	Fixed
QSP-2	Burning Shares when Claiming	Medium	Fixed
QSP-3	Possible to Lock Funds to the Zero Address	Low	Fixed
QSP-4	Rewards Are Locked if There Are No Shares	Low	Acknowledged
QSP-5	Overflows/Underflows Cause Claiming of Excessive Rewards	High	Fixed
QSP-6	Application Monitoring Can Be Improved by Emitting More Events	Informational	Fixed
QSP-7	Reduce Complexity	Informational	Fixed
QSP-8	Using Custom Math Code	Informational	Acknowledged
QSP-9	Unbounded Iteration	Informational	Acknowledged
QSP-10	Block Timestamp Manipulation	Informational	Acknowledged
QSP-11	Any User Can Claim on Behalf of Others	? Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

DISCLAIMER:
This audit focused exclusively on the `contracts/RewardsManager.sol` file.

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 One Vault Can Steal All Rewards

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/RewardsManager.sol`

Related Issue(s): SWC-101

Description: The `addBulkRewards` function allows a user to add distinct reward amounts to multiple epochs. The total reward amount is calculated in the following for-loop taken from `L559` to `L564`:

```
for(uint256 i; i < totalEpochs; ) {
    unchecked {
        total += amounts[i];
        ++i;
    }
}
```

If a user provides an `amounts` array `[2**256-1, 1]` as input to the `addBulkRewards` function, the `total` variable would be set to `0` due to an overflow which occurs within the for-loop on `L561`. Furthermore, no tokens would be added to the contract, since we use the `total` value in the token transfer call on `L568`. Despite not adding any reward tokens to the contract, we would still set `rewards[epochId][vault][token]` to `2**256-1` at `L576`. Any users with shares for this vault could then claim a huge amount of rewards, effectively stealing reward tokens which belong to other vaults.

Exploit Scenario:

1. The attacker deploys a generic vault which integrates with the `RewardsManager` contract in the standard way.
2. The attacker calls `addBulkRewards` with the following parameters
   - `startEpoch` = `1`
   - `endEpoch` = `2`
   - `vault` = `<vaultAddress>`
   - `token` = `<usdcAddress>`
   - `amounts` = `[2**256-1, 1]`
3. The attacker deposits assets into their vault, causing the vault to call `notifyTransfer`. This will give the attacker some shares.
4. After the epoch with an `epochId` of `1` has ended, the attacker claims the full USDC balance of the `RewardsManager` contract using the `claimReward` function.

Recommendation: Move `L561:total += amounts[i];` outside of the `unchecked` block.

Update: The Badger team has fixed the issue by removing the unchecked block surrounding the overflowing calculation.

QSP-2 Burning Shares when Claiming

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `contracts/RewardsManager.sol`

Description: The `claimBulkTokensOverMultipleEpochsOptimizedWithoutStorage` and `claimBulkTokensOverMultipleEpochsOptimizedWithoutStorageNonEmitting` functions both delete shares for the user at the provided `epochStart` in the `OptimizedClaimParams` struct. This happens on `L1121` and `L1222`. If either of these functions is called when `startEpoch == endEpoch`, then all the user's shares will be burned. This will prevent the user from claiming any further rewards to which they should have been entitled. It could also prevent a user from withdrawing their assets from a vault if that vault relies on the `notifyTransfer` function succeeding.

Recommendation: Consider restricting `claimBulkTokensOverMultipleEpochsOptimizedWithoutStorage` and `claimBulkTokensOverMultipleEpochsOptimizedWithoutStorageNonEmitting` so that `startEpoch` cannot equal `endEpoch`. An alternative would be to not delete shares when `startEpoch == endEpoch`.

Update: The Badger team has fixed the issue by setting the shares for `epochEnd` after deleting the shares for `epochStart`.

QSP-3 Possible to Lock Funds to the Zero Address

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/RewardsManager.sol`

Description: It is possible to add rewards to a vault with the zero address using the functions listed below:

- `L513:addBulkRewardsLinearly`
- `L548:addBulkRewards`
- `L586:addRewards`
- `L602:addReward`

Calling these functions by passing the zero address as the value for the `vault` parameter would make it impossible for those rewards to be claimed.

Recommendation: Validate that the `vault` parameter is not the zero address when calling functions to add rewards.

Update: The Badger team has fixed the issue by adding the recommended validation checks for the zero address.

QSP-4 Rewards Are Locked if There Are No Shares

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/RewardsManager.sol`

Description: If rewards are added to a vault for a particular epoch and no one has held any assets in that vault for that epoch, the rewards will be locked in the `RewardsManager` contract forever. This is because no one will have points or shares in that epoch for that vault. It can be seen in `L300` below that rewards paid out are based on a user's points for a particular epoch:

```
uint256 tokensForUser = totalAdditionalReward * userInfo.userEpochTotalPoints / (vaultInfo.vaultEpochTotalPoints - thisContractInfo.userEpochTotalPoints);
```

Recommendation: Consider adding a function to withdraw rewards when the total supply and total points for the vault are zero in a past epoch.

Update: The Badger team has indicated that this case is unlikely to occur and it is not worth introducing more complexity by adding a clawback function.

QSP-5 Overflows/Underflows Cause Claiming of Excessive Rewards

Severity: *High Risk*

Status: Fixed

File(s) affected: `contracts/RewardsManager.sol`

Related Issue(s): SWC-101

Description: There are multiple instances in the code where overflows or underflows could occur:

- `L111:totalPoints[epochId][vault] += timeLeftToAccrue * supply;` when the `supply` is very large.
- `L135:return maxTime - epochData.startTimestamp;` when `maxTime` is less than `epochData.startTimestamp`.
- `L658:totalSupply[cachedCurrentEpoch][vault] += amount;` when `totalSupply[cachedCurrentEpoch][vault]` is very large.
- `L843:uint256 start = DEPLOY_TIME + SECONDS_PER_EPOCH * (epochId - 1);` when the `epochId` is very large.

A particularly dangerous scenario would occur when a malicious vault causes an overflow in `totalSupply[cachedCurrentEpoch][vault]` at `L658`, but does not cause an overflow in `shares[cachedCurrentEpoch][vault][to]` at `L655`. This could result in the user's shares being greater than the vault's total shares, thereby allowing the user to claim more than the vault's total rewards on `L334:uint256 tokensForUser = totalAdditionalReward * userInfo.userEpochTotalPoints / vaultInfo.vaultEpochTotalPoints;`

Exploit Scenario:

1. The attacker adds a small amount of USDC rewards for a new malicious vault for the current epoch.
2. The malicious vault calls `notifyTransfer(address(0), addressX, 2)`. The state of the contract at this point is as follows:
   - `shares[currentEpoch][vault][addressX]` is `2`.
   - `totalSupply[currentEpoch][vault]` is `2`.
3. The malicious vault calls `notifyTransfer(address(0), addressY, 2^256 - 1)`. The state of the contract at this point is as follows:
   - `shares[currentEpoch][vault][addressX]` is `2`.
   - `shares[currentEpoch][vault][addressY]` is `2^256 - 1`.
   - `totalSupply[currentEpoch][vault]` is `1` due to an overflow on `L658`.
   - The shares of `addressX` are now double the `totalSupply` of the vault.
4. Once the `currentEpoch` has passed, the attacker calls `claimRewardReference(epoch, vault, usdcAddress, addressX)`. The vault and user will both get accrued during the `claimRewardsReference` call. However, the points for the user will get accrued twice as fast as the total points for the vault since the user has twice the amount of shares as the vault's totalSupply. This would lead the `userPoints/vaultPoints` ratio to become greater than `1`.
5. The attacker can call any function for claiming rewards using `addressX` to claim more than the vault's share of rewards.

Recommendation: Move `L111`, `L121`, `L658` and `L841` outside of `unchecked` blocks.

Update: The Badger team has fixed the issue by moving the unchecked statements related to the total supply outside of the unchecked blocks. They have also split the epoch-related functions into `internal` and `external` versions, such that the `external` versions will revert on overflows. Although the `internal` versions of these functions could technically still overflow, this should not happen since related validation for input epochs are present wherever these `internal` functions are referenced.

QSP-6 Application Monitoring Can Be Improved by Emitting More Events

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/RewardsManager.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 transition can be logged, which is beneficial for monitoring the contract and also tracking eventual bugs, or hacks. Emitting events also allows off-chain applications to dynamically respond to important state transitions. Below we present a non-exhaustive list of events that could be emitted to improve the application management:

- `AddRewards`
- `ClaimRewards`
- `Transfer`
- `UserAccrual`
- `VaultAccrual`

Recommendation: Emit the suggested events during important state transitions.

Update: The Badger team has added multiple events for tracking state changes.

QSP-7 Reduce Complexity

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/RewardsManager.sol`

Description: There are currently many ways to claim rewards, each with different ways of preventing duplicate claims. It may not be clear to users which functions to use when claiming rewards. Some functions update `pointsWithdrawn` so that points are not withdrawn a second time, and others delete the shares and points from storage so that reward calculations return zero.

Recommendation: If a reward-claiming function is less gas efficient and offers no unique benefits, it can be removed to reduce complexity.

Update: The complexity of the contract has been reduced by removing the functions `claimBulkTokensOverMultipleEpochsOptimized` and `addRewards`.

QSP-8 Using Custom Math Code

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/RewardsManager.sol`

Description: The contract implements its own `_min` function. Furthermore, some parts of the code reproduce a minimum function in a different way such as in `L750` to `L754` below:

```
// Cap maxTime at epoch end
uint256 maxTime = block.timestamp;
if(maxTime > epochData.endTimestamp) {
    maxTime = epochData.endTimestamp;
}
```

It would be better to use a common audited library such OpenZeppelin's Math.sol which contains `min` and `max` functions.

Recommendation: Replace the custom `_min` function by using the `min` function from OpenZeppelin's Math.sol library. Also, try to use the `min` function instead of recreating the functionality for example from `L750` to `L754`.

Update: The Badger team chose to copy the `_min` function from the library directly rather than importing the full library. They have also replaced the repetitive code referenced in the issue by using the `_min` function.

QSP-9 Unbounded Iteration

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/RewardsManager.sol`

Description: Multiple functions take arrays as parameters and iterate over these arrays. These functions do not enforce an upper bound on the length of the arrays. If the array is long enough, the function could fail due to the gas usage being too high. We have listed all such functions below:

- `L203:claimRewards`
- `L233:claimRewardReference`
- `L348:claimBulkTokensOverMultipleEpochs`
- `L415:claimBulkTokensOverMultipleEpochsOptimized`
- `L513;addBulkRewardsLinearly`
- `L548:addBulkRewards`
- `L586:addRewards`
- `L1041:claimBulkTokensOverMultipleEpochsOptimizedWithoutStorage`
- `L1139:claimBulkTokensOverMultipleEpochsOptimizedWithoutStorageNonEmitting`

Recommendation: Consider implementing an upper bound for input array length.

Update: The Badger team indicated that they would rather have the caller choose how to handle the risk over putting an arbitrary limitation such as an upper bound. Large inputs can be chunked into multiple calls so that the user does not run out of gas.

QSP-10 Block Timestamp Manipulation

Severity: Informational

Status: Acknowledged

File(s) affected: `contracts/RewardsManager.sol`

Related Issue(s): SWC-116

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account.

The `RewardsManager` contract uses timestamps in many ways including determining the current epoch and calculating the time since the last accrual. Users should be aware that these calculations can be affected by the deviation of `block.timestamp` from the clock time.

Recommendation: While we did not find any particularly problematic cases, the limitations of `block.timestamp` should be taken into account.

Update: The Badger team has indicated that besides minor accrual differences, there should not be any significant impact of block timestamp manipulation.

QSP-11 Any User Can Claim on Behalf of Others

Severity: Undetermined

Status: Acknowledged

File(s) affected: `contracts/RewardsManager.sol`

Description: `claimRewards`, `claimRewardReference`, `claimReward`, `claimRewardNonEmitting` and `claimBulkTokensOverMultipleEpochs` allow anyone to claim on behalf of a user. This could result in a user receiving their reward tokens before they want to or expect to receive them. This could cause confusion for a user who may be trying to reclaim their reward tokens when their rewards have already been claimed.

Recommendation: Only allow a user to claim their own reward tokens.

Update: For now, the Badger team wishes to keep this design to save on gas costs for users. Before launching, the team may reconsider changing this design.

Automated Analyses

Slither

Slither found 71 issues most of which were false positives. The remaining issues have been included in this report.

Code Documentation

1. Several mappings are documented with code comments. However, the code comments come after the mapping declarations which makes them hard to read. We recommend moving the code comment above the mapping declaration for better readability. (**Update:** Fixed)

2. The code comment at `L188` should say `lastAccruedTimestamp` instead of `lastUserAccrueTimestamp`. (**Update:** Fixed)

3. Although most functions are documented to some extent using the NatSpec standard, they are missing NatSpec documentation for parameters. We recommend using the `@param` tag to document the parameters of every `external` and `public` function. (**Update:** Fixed)

4. The code comment at `L573` incorrectly states `Give each epoch an equal amount of reward.` In actuality, each epoch is assigned an amount based on the `amounts` array given to the function. (**Update:** Fixed)

5. It is not clear what the `// ===== EXPERIMENTAL ==== ////` section of the code refers to. It looks as if the functions in that section have been removed. Consider removing the `// ===== EXPERIMENTAL ==== ////` section's code comments from `L878` to `L898`. (**Update:** Fixed)

6. There is a typo on `L454` where `maintaining` is spelled as `maintainingn`. (**Update:** Fixed)

Adherence to Best Practices

1. Consider replacing the hardcoded `604800` value on `L57` with an arithmetic expression such as `7*24*60*60`. This would make it more clear that it refers to the number of seconds in a week.
2. All `require` statements lack error messages. Add descriptive and clear error messages for better user experience and ease of troubleshooting. (**Update:** Fixed)
3. The `epochs` function on `L851` is redundant as the `getEpochData(uint256 epochId)` function provides the same functionality. Consider removing the `epochs` function. (**Update:** Fixed)
4. Some lines of code are very long such as `L923`. Consider limiting the length of a line of code to `79` or `99` characters as recommended by the Solidity style guide. (**Update:** Acknowledged)
5. Some function names are quite long such as `claimBulkTokensOverMultipleEpochsOptimizedWithoutStorage` and `claimBulkTokensOverMultipleEpochsOptimizedWithoutStorageNonEmitting`. Consider shortening the names of functions to improve readability. (**Update:** Fixed)
6. There are hanging TODO's on `L926` and `L986`. (**Update:** Fixed)
7. Mark `getUserNextEpochInfo` and `getVaultNextEpochInfo` as `private`, since they may return incorrect data if passed the wrong previous epoch supply/balance.
8. Mark `getVaultTimeLeftToAccrue` as `private` or require that the `epochId` not be in the future. (**Update:** Fixed)
9. The contract contains a lot of comments that explain design decisions, these should be consolidated into the README or architecture diagrams. (**Update:** Acknowledged)
10. The functions `claimRewardNonEmitting` and `claimBulkTokensOverMultipleEpochsOptimizedWithoutStorageNonEmitting` are designed for non-emitting vaults and should not be used when that is not the case. Consider preventing the misuse of these functions by modifying the code or by educating the users to be aware of that. (**Update:** Fixed)
11. Consider adding a `nonReentrant` modifier to the following functions: (**Update:** Acknowledged but not fixed due to composability concerns.)
    1. `L203:claimReward`
    2. `L309:claimRewardNonEmitting`
    3. `L233:claimRewardReference`
    4. `L348:claimBulkTokensOverMultipleEpochs`
    5. `L415:claimBulkTokensOverMultipleEpochsOptimized`
    6. `L1041:claimBulkTokensOverMultipleEpochsOptimizedWithoutStorage`
    7. `L1139:claimBulkTokensOverMultipleEpochsOptimizedWithoutStorageNonEmitting`

Test Results

Test Suite Results

```
tests/attacks/test_balance_in_future_reward_steal.py ..                                             [  2%]
tests/attacks/test_bug_without_storage_looses_shares.py .                                           [  3%]
tests/attacks/test_bulk_claim_duplicate_tokens.py .                                                 [  4%]
tests/attacks/test_overflow_risk.py ..                                                              [  6%]
tests/integration/test_basic_integration.py .....                                                   [ 12%]
tests/lens/test_lens_is_equivalent.py ...                                                           [ 15%]
tests/lifecycle/test_basic_set_earn_claim_flow.py .                                                 [ 16%]
tests/lifecycle/test_no_storage_claim.py ...                                                        [ 20%]
tests/lifecycle/test_reverts.py .......                                                             [ 28%]
tests/unit/test_accrue_points.py ...                                                                [ 31%]
tests/unit/test_add_reward.py ...                                                                   [ 34%]
tests/unit/test_add_rewards_bulk.py .....                                                           [ 40%]
tests/unit/test_add_rewards_bulk_linear.py ......                                                   [ 47%]
tests/unit/test_bulk_claim_rewards.py .....                                                         [ 52%]
tests/unit/test_bulk_claim_rewards_optimize_no_storage.py ......                                    [ 59%]
tests/unit/test_bulk_claim_rewards_optimize_no_storage_non_emitting.py ......                       [ 66%]
tests/unit/test_deposit_balance_tracking.py ....                                                    [ 70%]
tests/unit/test_epoch_start.py ..                                                                   [ 73%]
tests/unit/test_non_accrual_is_equivalent.py .                                                      [ 74%]
tests/unit/test_notify_transfer.py .......                                                          [ 82%]
tests/unit/test_points_claim_rewards.py ....                                                        [ 86%]
tests/unit/test_time_to_accrue_math.py ......                                                       [ 93%]
tests/unit/test_user_next_epoch_info.py ..                                                          [ 95%]
tests/unit/test_vault_next_epoch_info.py ..                                                         [ 97%]
tests/unit/test_weird.py ..                                                                         [100%]

================================================================================ 89 passed in 33.58s ================================================================================
```

Code Coverage

The code coverage results were generated by running the `brownie test --coverage` command.

```
contract: RewardsManager - 99.4%
RewardsManager._getBalanceAtEpoch - 100.0%
RewardsManager._getTotalSupplyAtEpoch - 100.0%
RewardsManager._getUserNextEpochInfo - 100.0%
RewardsManager._getUserTimeLeftToAccrue - 100.0%
RewardsManager._getVaultNextEpochInfo - 100.0%
RewardsManager._getVaultTimeLeftToAccrue - 100.0%
RewardsManager._min - 100.0%
RewardsManager._requireNoDuplicates - 100.0%
RewardsManager.accrueUser - 100.0%
RewardsManager.accrueVault - 100.0%
RewardsManager.addBulkRewards - 100.0%
RewardsManager.addBulkRewardsLinearly - 100.0%
RewardsManager.claimBulkTokensOverMultipleEpochs - 100.0%
RewardsManager.claimReward - 100.0%
RewardsManager.claimRewardNonEmitting - 100.0%
RewardsManager.claimRewardReference - 100.0%
RewardsManager.claimRewards - 100.0%
RewardsManager.getBalanceAtEpoch - 100.0%
RewardsManager.getClaimableBulkRewards - 100.0%
RewardsManager.getTotalSupplyAtEpoch - 100.0%
RewardsManager.getUserNextEpochInfo - 100.0%
RewardsManager.getUserTimeLeftToAccrue - 100.0%
RewardsManager.getVaultNextEpochInfo - 100.0%
RewardsManager.getVaultTimeLeftToAccrue - 100.0%
RewardsManager.notifyTransfer - 100.0%
RewardsManager.reap - 100.0%
RewardsManager.tear - 100.0%
RewardsManager.addReward - 50.0%
```

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

```
49fa0f02abae8c8a0f1119c4a4e82c09e23743cedc2fdd553951382c4641ab32 ./contracts/RewardsManager.sol
```

Tests

```
ef471c4fe2eb629921ebbe98b6ad6df316afa94ea85f2bc96f3ed38c34d74f0a ./tests/conftest.py
2969311c5b664e854338fc43581dc22e12c3d354953a0020249aa5acf47b2ac7 ./tests/attacks/test_bulk_claim_duplicate_tokens.py
bc43423bd43ce7f337fc238d57de52d7623a1a7e341d0b2be2dcdc1300a35bdb ./tests/attacks/test_balance_in_future_reward_steal.py
643ec80e42223d5661ac0df5f16df73c65be54745a37c352113f9d544872bd49 ./tests/attacks/test_overflow_risk.py
bba16a936f545736ddb9bfbcd2be373cf23e25909869dac3e4b9d0ff2edbf6ea ./tests/attacks/test_bug_without_storage_looses_shares.py
537213ea33ed23f60be2e2525b03c56976a6adfe8bdb16359829b01be0997f15 ./tests/lens/test_lens_is_equivalent.py
ecd3e2742327021d3b5f41fb848066382b632f0dc2b74cfa2f544435675739d7 ./tests/simulation/one_year_claim_vs_accrual.py
2789a8d8caeca758be0c911c106dd028877e2e1794674c910a2f628112ec2391 ./tests/simulation/basic_deposit_transfer_gas.py
8c523220fef0d91d43873461dd570d882f58b32b3f3a176e7f3fca353816934e ./tests/simulation/one_week_of_accrual.py
0d484590b2299c7ed7baa42f507b18dd015c488917ecc9f2ea1c69daaadad030 ./tests/simulation/one_year_of_accrual.py
3ca01cbaddf1d1a3ca80ec282111c8077b9d49f82e647d65c26c2ab81a538683 ./tests/lifecycle/test_reverts.py
f09c3a09b1f09860e89715949ff7e3972b3f83ec459b5848d4cf6db2adfaa5bf ./tests/lifecycle/test_no_storage_claim.py
5db24072f1d61f5c611418326b501b13a78cd03dd556b52f170f49542e275a06 ./tests/lifecycle/test_basic_set_earn_claim_flow.py
e899888e2436176c424be8d518b384caa2507fb06e95362a095b9dd9e47e321b ./tests/integration/conftest.py
94385fa3f335f18554a173b0a33563ac3c9641113b70b526da1fe6809c1ef42a ./tests/integration/test_basic_integration.py
929b298ec7b0473e551e267da95b15024ee05075d70ed0b26716a64457a7766b ./tests/unit/test_epoch_start.py
e9ea1f1557c697e44c99f6c878fefed6bdd57461135fbb8beaf1bc9782cd4e66 ./tests/unit/test_accrue_points.py
a226a13bd4c4e8a77b6dca30b1764a3eea7909c18be8c4fb0d6054822f352b8c ./tests/unit/test_bulk_claim_rewards.py
4602f3e4bf404c9ca0c92acd2e678c9ddbff201f39ebab6f7e049b7284c0012f ./tests/unit/test_add_rewards_bulk.py
832fb85d337d59abfe530c49f2e9f611a0b04224c1ef2530c4f0a80a13a4f266 ./tests/unit/test_weird.py
91b1c2ba523532bce191b395276d33e8e6c6a9e87a8c154cc6ecd9a0fde3472c ./tests/unit/test_bulk_claim_rewards_optimize_no_storage.py
6e025b8ff22bbfa5abf9376d0ec17694b3ce3e167d1f6624128c8914c3eefd2d ./tests/unit/test_add_rewards_bulk_linear.py
1d76051bb66d12aa22c54127ff7f4afaa7de0f873b284e5942b53ad5fcf7f204 ./tests/unit/test_user_next_epoch_info.py
58a28e5bb5c29670907f6bd19ab4862b59b3dbea4f0a570a81cc2b36af64a512 ./tests/unit/test_vault_next_epoch_info.py
c7d41e5188a67b08a409c36d03e2f482256d38bda96ccf35add90cdf18838c6d ./tests/unit/test_time_to_accrue_math.py
ce83a9a5a2f7f71925e158df3b4003eb32c00c662830cb813b12f29a9236f1b8 ./tests/unit/test_deposit_balance_tracking.py
7ebd0e3739e79843a27a3c7ff9522b3ed965e6903a88f96d01a520052a348773 ./tests/unit/test_points_claim_rewards.py
7867a476e68330654cff0036879351783c563d139fba12ef278937a565ec2438 ./tests/unit/test_notify_transfer.py
7af7af17bdfac643bb7e66faf6e8ca86982b74b67405af1214f9a85ce8c844f9 ./tests/unit/test_bulk_claim_rewards_optimize_no_storage_non_emitting.py
52c31aea3f1e8c82f4a60986a585449ffd7459b686be15feacf7b833f69e867a ./tests/unit/test_non_accrual_is_equivalent.py
d5416a0ba63396f548e1245a8813eaf6416815da0054478b3789672c9278806b ./tests/unit/test_add_reward.py
```

Changelog

- 2022-07-12 - Initial report
- 2022-08-03 - Final report

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.