Quantstamp Security Assessment Certificate

January 8th 2022 — Quantstamp Verified

Vault.Inc

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 Poming Lee, Research Engineer
Timeline	2021-12-13 through 2022-01-07
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review.
Specification	None
Documentation Quality	Low
Test Quality	High

Source Code

Repository	Commit
vault-tec-core	a0cd3ec

Total Issues	8 (6 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	1 (1 Resolved)
Low Risk Issues	4 (2 Resolved)
Informational Risk Issues	1 (1 Resolved)
Undetermined Risk Issues	1 (1 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 Vault.Inc "vault-tec-core" repository. Quantstamp found several issues. Some issues were unavoidable due to the design of the system, while others were fixed. All issues have been resolved or acknowledged. The code is accompanied by tests with fairly high coverage.

ID	Description	Severity	Status
QSP-1	Critically Low Test Coverage	High	Fixed
QSP-2	Maximum Approve	Medium	Mitigated
QSP-3	Privileged Roles and Ownership	Low	Acknowledged
QSP-4	Unknown Code in the Contract	Low	Acknowledged
QSP-5	Use of Insecure Casting Operation	Low	Fixed
QSP-6	Missing Input Validation	Low	Fixed
QSP-7	Events Not Emitted on State Change	Informational	Fixed
QSP-8	Potential Re-Entrancy	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 Critically Low Test Coverage

Severity: *High Risk*

Status: Fixed

File(s) affected: `all`

Description: There are no tests, and as such, the test coverage is 0%.

Recommendation: Add (many) tests to increase coverage to the highest possible amount: as close to 100% coverage as possible.

Update: The team has added tests.

QSP-2 Maximum Approve

Severity: *Medium Risk*

Status: Mitigated

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 partially mitigated by leaving the “unlimited” approval, but resetting it to zero when removing the pool.

QSP-3 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: The team has acknowledged this issue: "We will be adding a note in the frontend regarding the admin role.".

QSP-4 Unknown Code in the Contract

Severity: *Low Risk*

Status: Acknowledged

Description: The constructor of `contracts\base\AbstractRewards.sol` utilizes code unknown (variables of function type) during the audit period to initialize the implementation of two functions, that are: `getSharesOf` and `getTotalShares`. This could cause the contracts to result in very different behavior from what was expected.

Recommendation: Hard code the intended logic of those two functions into the contract and have them audited. Or provide comments that explain why this is intended and should not be changed.

Update: The team has acknowledged this issue: "The calculation of reward distribution requires the ERC20 functions and we don’t want to add that dependency to AbstractRewards contract. It’s only called in BasePool’s constructor and we always pass in the standard ERC20 functions as the parameters so we think the behavior is predictable."

QSP-5 Use of Insecure Casting Operation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AbstractRewards.sol`

Related Issue(s): SWC-101

Description: In `AbstractRewards._correctPoints()` the insecure primitive casting operation `int256()` is used. For sufficiently large positive or negative values this cast may wrap around, without leading to a revert and therefore lead to unexpected behaviour.

Recommendation: Replace the use of this insecure cast operation with for example its secure counterpart `.toInt256()` of OpenZeppelins SafeCast library.

Update: This has been resolved by changing the primitive cast operation to its safe counterpart `.toInt256()` of OpenZeppelins `SafeCast` library, as suggested.

QSP-6 Missing Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AbstractRewards.sol`, `TimeLockPool.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. `AbstractRewards._prepareCollect()` does not check that parameter `_account` is different from `address(0)`.
2. `AbstractRewards._correctPointsForTransfer()` does not check that parameters `_from` and `_to` are different from `address(0)` or `_shares` is non-zero.
3. `AbstractRewards._correctPoints()` does not check that parameter `_account` is different from `address(0)` or `_shares` is non-zero.
4. `TimeLockPool.deposit()` does not check that parameter `_receiver` is different from `address(0)`.
5. `TimeLockPool.withdraw()` does not check that parameter `_receiver` is different from `address(0)`.

Recommendation: Add corresponding checks on the listed parameters, i.e. via `require` statements.

Update: This issue has been resolved, by adding corresponding parameter checks, as suggested.

QSP-7 Events Not Emitted on State Change

Severity: *Informational*

Status: Fixed

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 resolved by adding a new event `PointsCorrectionUpdated` and emitting it where needed, as suggested

QSP-8 Potential Re-Entrancy

Severity: *Undetermined*

Status: Fixed

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

Related Issue(s): SWC-107

Description: The following functions do not follow the `Checks-Effects-Interactions` pattern, as they perform calls to external contracts before changing state variables, while at the same time not being protected through i.e. a `nonReentrant` modifier:

- `TimeLockPool.deposit()`
- `BasePool.distributeRewards()`

Note that a re-entrancy would be also possible for seemingly benign `rewardToken` or `depositToken` tokens, in case those are based on the ERC777 standard, allowing to hook transfers and divert control flow.

Recommendation: Use the OpenZeppelin ReentrancyGuard.nonReentrant modifier, as was already used in i.e. `Vault._deposit()`.

Update: This issue has been resolved by making use of the `nonReentrant` modifier at said functions, as suggested.

Automated Analyses

Slither

Slither's results were filtered and either added in the report, or omitted as false positives.

Code Documentation

1. The `require` error message in line 48 of `LiquidityMiningManager.sol` states `rewardSource token must be set`, which seems to be a copy-and-paste of the previous line and should have been `rewardSource address must be set` as it is seems to be a non-token address.

Adherence to Best Practices

1. For improved readability it is recommended to have a maximum line length of 79 or 99. Therefore L31, L36, L47, L48, L55, L57, L91 and L131 of `LiquidityMiningManager.sol`, L17 of `TimeLockNonTransferablePool.sol`, L36, L37, L42, L43, L67, L71, L72 and L75 of `TimeLockPool.sol`, L53 and L59 of `AbstractRewards.sol`, L26 and L84 of `BasePool.sol` and L13, L16 and L24 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. For clarity and consistency magic numbers should be declared once, commented and then used throughout. In this regard the number `1e18` is used in L60, L72 and L87 of `TimeLockPool.sol` and L37 and L72 of `BasePool.sol`, without being declared or commented. To conform to best practices consider declaring this constant, i.e. in `BasePool.sol` and comment it.

4. `contracts\base\BasePool.sol`, line 41: Consider checking that if `_escrowPortion > 0`, should revert if `_escrowPool == 0x0`.

5. According to best practices address parameters of events should always be indexed to facilitate logging and monitoring. The address parameter `_from` in L44 of `LiquidityMiningManager.sol` is however lacking the `indexed` keyword and should therefore be added.

6. `LiquidityMiningManager`: `MAX_POOL_COUNT` could be made constant.

7. `LiquidityMiningManager.distributeRewards()` ignores return value by `address(pool.poolContract).call()` (Line 134). This is noted as intentional in a comment, but perhaps should be handled in the code itself.

Test Results

Test Suite Results

```
BasePool
  distributeRewards
     ✓ Should fail when there are no shares
     ✓ Should fail when tokens are not approved (221ms)
     ✓ Should work (497ms)
  claimRewards
     ✓ First claim single holder (601ms)
     ✓ Claim multiple holders (940ms)
     ✓ Multiple claims, distribution and holders (1301ms)
     ✓ Zero escrow (289ms)
     ✓ Full escrow (792ms)

LiquidityMiningManager
  Adding pools
     ✓ Adding a single pool (46ms)
     ✓ Adding multiple pools (87ms)
     ✓ Adding a pool twice should fail (39ms)
     ✓ Adding a pool from a non gov address should fail
  Removing pools
     ✓ Removing last pool in list (104ms)
     ✓ Removing a pool in the beginning of the list (162ms)
     ✓ Removing all pools (786ms)
     ✓ 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 (1344ms)
     ✓ Should work (1073ms)
  Adjusting weight
     ✓ Adjust weight up (59ms)
     ✓ Adjust weight down (213ms)
     ✓ Should fail from non gov address
  Setting reward per second
     ✓ Should work (53ms)
     ✓ Should fail from non gov address

TimeLockNonTransferablePool
   ✓ transfer
   ✓ transferFrom

TimeLockPool
  deposit
     ✓ Depositing with no lock should lock it for 10 minutes to prevent flashloans (390ms)
     ✓ Deposit with no lock (378ms)
     ✓ Trying to lock for longer than max duration should lock for max duration (297ms)
     ✓ Multiple deposits (522ms)
     ✓ Should fail when transfer fails (73ms)
  withdraw
     ✓ Withdraw before expiry should fail
     ✓ Should work (238ms)

TokenSaver
  saveToken
     ✓ Should fail when called fron non whitelised address
     ✓ Should work (245ms)


36 passing (18s)
```

Code Coverage

File	Statements	Branches	Functions	Lines
contracts/	80.2% 81/101	66.67% 24/36	90% 18/20	80.58% 83/103
contracts/base/	82.46% 47/57	56.67% 17/30	88.24% 15/17	82.76% 48/58
contracts/interfaces/	100% 0/0	100% 0/0	100% 0/0	100% 0/0

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

```
757a1cc3ae7908d3bb5ba544d2d1cdafcb206fbf3a9c32c7097a9a3c057aecb7 ./contracts/LiquidityMiningManager.sol
65d66fa08c13c10901a59a4c3c19d9c1075396715491e8b4f48d659bf5dd77c5 ./contracts/TimeLockNonTransferablePool.sol
c30f39df9841ca892efefe97ff5b95af63fc4775ce10a02cef2a754a79600140 ./contracts/TimeLockPool.sol
c8c79af8a80a435b466588e5291d154bf31285aba868d30b0bc44326c1bcdaf9 ./contracts/View.sol
44129dc0fb197cdc38891d05da506588492794f89f6de83902a96f3b6d621281 ./contracts/test/TestBasePool.sol
34b79e96ba58a220965725b4f4c3b3350f06ef6330242b07e5ec80101cc20106 ./contracts/test/TestFaucetToken.sol
dc79250ac1a086a86e43daa8d7e5a0833f01013ea94298b933c1f6165b56872a ./contracts/test/TestToken.sol
e4d54710f7d465f7b264f10c571373c078dce11e2c677bf334220fcd97f68d0b ./contracts/interfaces/IAbstractRewards.sol
0a3671d77736ec30404857a50b6e7d3d4654beb58e3e3108b63b5360c309d2d9 ./contracts/interfaces/IBasePool.sol
cd806c0f1dc6637bb3e147b94e53d5af24089be02868f92152cc382f7431558d ./contracts/interfaces/ITimeLockPool.sol
8b3eb8c8026bdc84b7fea6601b73a188daca604aba2d5b81adce3f8da8295bff ./contracts/base/AbstractRewards.sol
f7363171e902f916f35b57a1c102890a8b7ae6b84aaf11129afc995c32ea4f34 ./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

- 2021-12-22 - Initial report [`b1c3e04`]
- 2022-01-07 - Revised report [`a0cd3ec`]

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.