Quantstamp Security Assessment Certificate

December 2nd 2021 — Quantstamp Verified

JPEG'd

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

Executive Summary

Type	NFT Lending
Auditors	Jose Ignacio Orlicki, Senior Engineer Cristiano Silva, Research Engineer Marius Guggenmos, Senior Research Engineer
Timeline	2021-10-25 through 2021-12-02
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Medium
Test Quality	High

Source Code

Repository	Commit
jpegd	d210f47
jpegd	679bb3c (reaudit)

Total Issues	30 (26 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	3 (3 Resolved)
Low Risk Issues	11 (10 Resolved)
Informational Risk Issues	14 (11 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

We have reviewed the code, documentation, and test suite and found several issues of various severities. Overall, we consider the code to be well-written but with insufficient documentation in the form of inline comments and docstrings. The test suite is very extensive but can be improved given the suggested changes from this report. We used also extra private documentation delivered to the audit team that was very detailed and included an architecture diagram. We have outlined suggestions to better follow best practices, and recommend addressing all the findings to tighten the contracts for future deployments or contract updates. We recommend addressing all the **32** findings to harden the contracts for future deployments or contract updates. We recommend against deploying the code as-is.
Update: Quantstamp has audited the changes based on the commit for the `jpegd` repository (679bb3c). Of the original 30 issues, all 30 have been either fixed (26) or acknowledged (4). Code documentation is very good, but we still consider public documentation&specs to be lacking.

ID	Description	Severity	Status
QSP-1	Unchecked Credit Limit in `ERC20VaultForDAO.borrow()`	High	Fixed
QSP-2	Division Before Multiplication In Pool Reward Calculation	Medium	Fixed
QSP-3	Unchecked Cast On External Data	Medium	Fixed
QSP-4	Missing Oracle Output Validation	Medium	Fixed
QSP-5	Integer Math Can Lead To Non-linear Vesting Schedule	Low	Fixed
QSP-6	Transferring PreJPEGD Tokens Can Break The Vesting Schedule	Low	Fixed
QSP-7	Share Dilution Concerns in LPFarming	Low	Fixed
QSP-8	Missing Input Validation	Low	Fixed
QSP-9	Cannot Repay Less Than The Interest Owed	Low	Fixed
QSP-10	Missing Validation In Credit Limit Rate	Low	Fixed
QSP-11	Absence Of Events Logging	Low	Fixed
QSP-12	Safe Transfer Is Not Fully Compatible	Low	Acknowledged
QSP-13	Ether Can Be Locked In With ERC20 Deposit	Low	Fixed
QSP-14	Allowing Operations With Zero Amounts	Low	Fixed
QSP-15	Minor `lockedTokens` View Calculation Error	Low	Fixed
QSP-16	`rewardDebt` Variable In `LPFarming` Can Be Simplified	Informational	Fixed
QSP-17	Complex Calculation Concerns In `_transfer`	Informational	Fixed
QSP-18	SafeMath Not Needed For Solidity Versions >= 0.8	Informational	Fixed
QSP-19	Use `immutable` For Members Only Changed In Constructor	Informational	Fixed
QSP-20	Unexplained Magic Numbers	Informational	Fixed
QSP-21	Inaccurate Name for ERC20VaultForDAO	Informational	Fixed
QSP-22	Code Duplication In `NFTVault.borrow`	Informational	Fixed
QSP-23	Gas Optimization In Mapping	Informational	Fixed
QSP-24	Renouncing The Ownership Is Possible	Informational	Fixed
QSP-25	Missing Docstrings And Inline Comments In The Code	Informational	Fixed
QSP-26	Unlocked Pragma	Informational	Acknowledged
QSP-27	Gas Optimization For Errors	Informational	Acknowledged
QSP-28	Contract Size Optimizations	Informational	Fixed
QSP-29	Time Period Can Be More Accurate	Informational	Acknowledged
QSP-30	Convoluted Unclaimed Transfer Semantics	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 v0.8.0

Steps taken to run the tools:

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

Findings

QSP-1 Unchecked Credit Limit in `ERC20VaultForDAO.borrow()`

Severity: *High Risk*

Status: Fixed

File(s) affected: `vaults/ERC20VaultForDAO.sol`

Description: The `borrow` function only checks whether the amount requested in the function call is within the credit limit. Repeatedly calling borrow with the credit limit allows infinite borrowing.

```
function borrow(uint256 amount) external onlyOwner nonReentrant {
    uint256 creditLimit = _getCreditLimit(collateralAmount);
    require(amount <= creditLimit, "insufficient_credit");

    // mint stablecoin
    IStableCoin(stablecoin).mint(msg.sender, amount);

    // update position
    debtAmount += amount;
}
```

Recommendation: Require that the global `debtAmount` is less than the credit limit. The developers should check if this is the intended behavior. Apparently, the line 102 `require(amount <= creditLimit, "insufficient_credit")` should be changed to `require(amount + debtAmount <= creditLimit, "insufficient_credit")`.

Update: Fixed in this PR.

QSP-2 Division Before Multiplication In Pool Reward Calculation

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `farming/LPFarming.sol`

Description: The `_updatePool` function calculates the rewards by taking the block delta since the last update and multiplying it by the pool allocation relative to the total allocation.

```
uint256 reward = ((block.number - pool.lastRewardBlock) *
    rewardPerBlock *
    pool.allocPoint) / totalAllocPoint;
pool.accRewardPerShare =
    pool.accRewardPerShare +
    (reward * (1e36)) /
    lpSupply;
```

Because the division by `totalAllocPoint` takes place before the multiplication by `1e36`, the reward value might be truncated due to integer division.
This issue is more pronounced with a higher total allocation point and more frequent updates. This error accumulates for every single update, which can lead to a significant deviation from the promised tokens.

Exploit Scenario: We discuss a numerical example.

```
block.number = 50
pool.lastRewardBlock = 41
rewardPerBlock = 2
pool.allocPoint = 5
totalAllocPoint = 100

((50 - 41) * 2 * 5) / 100 =
90 / 100 = 0
```

While this is an extreme example where the rewards end up 0, the same happens if the calculation ends up at `190 / 100`. It basically always truncates the result, which ends up hurting the liquidity providers.

Recommendation: Since the reward is multiplied by `1e36` afterwards anyway, perform the multiplication before the division by `totalAllocPoint` like this

```
uint256 reward = ((block.number - pool.lastRewardBlock) *
    rewardPerBlock * 1e36
    pool.allocPoint) / totalAllocPoint;
pool.accRewardPerShare =
    pool.accRewardPerShare +
    reward / lpSupply;
```

Update: Fixed in this PR.

QSP-3 Unchecked Cast On External Data

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `vaults/ERC20VaultForDAO.sol`

Description: In the following function, a price is retrieved from an oracle and being cast from signed to unsigned. In case the value is negative, this would lead to a huge price for the collateral.

```
function _collateralPriceUsd() internal view returns (uint256) {
    (, int256 answer, , , ) = oracle
        .latestRoundData();

    return uint256(answer) * 10**10;
}
```

Recommendation: Either check that the integer that has been returned is positive or (and) put a link to the documentation of the oracle that shows the value can never be negative.

Update: Fixed in this PR.

QSP-4 Missing Oracle Output Validation

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `ERC20VaultForDAO.sol`, `ERC20VaultForDao.sol`

Description: Since Oracle is an external entity, the code must be defensive against issues that may arise in this external entity. Below we present two functions that assume the return from the Oracle without any validation.

```
function _collateralPriceUsd() internal view returns (uint256) {
    (, int256 answer, , , ) = IAggregatorV3Interface(oracle)
        .latestRoundData();

    return uint256(answer) * 10**10;
}
```

```
function _updateOracle() internal {
    IOracle(oracle).update();
    ethPriceUSD = IOracle(oracle).eth_usd_18();
    jpegPriceUSD = IOracle(oracle).jpeg_usd_18();
}
```

Recommendation: Check if the returned price is non-zero. Be aware that querying the price of token A x B is not the same as requesting the price of token B in terms of A.

Update: Fixed in this PR.

QSP-5 Integer Math Can Lead To Non-linear Vesting Schedule

Severity: *Low Risk*

Status: Fixed

File(s) affected: `vesting/PreJPEGD.sol`

Description: The calculation for how many tokens per second are vesting is done by calculating
`tokensPerSecond = totalTokens / (endTime - startTime)`
Since this is done using integer math, the result will be truncated. This can lead to scenarios where a significant amount of the tokens are only vested once the the vesting period is over, originating from the following equation:
`totalTokens = tokensPerSecond * (endTime - startTime) + TruncationError`

Exploit Scenario: - vesting period (`endTime - startTime`) is 1000 seconds

- a user owns 9999 tokens
- `tokensPerSecond = 9999 / 1000 = 9`, so 9 tokens per second will be vesting
- `TruncationError = 999`
- At the end of the vesting period, only 9000 tokens will have vested (`9tps * 1000`)
- Once the vesting period is over, the full balance can be claimed regardless of the tokens per second
- remaining 999 tokens all vest at once (around 10% of the tokens in that case)

Recommendation: Either choose one of the following alternatives.

- acknowledge and document this behavior.
- only keep track of the amount of tokens that have been claimed already and compute vesting each time a user claims their tokens.
- use additional precision in the computation.

Update: According to the dev team "Requirements for PreJPEG changed and the token isn't transferrable anymore.". Fixed in this PR.

QSP-6 Transferring PreJPEGD Tokens Can Break The Vesting Schedule

Severity: *Low Risk*

Status: Fixed

File(s) affected: `vesting/PreJPEGD.sol`

Description: The `_transfer` function in `PreJPEGD` uses integer math to calculate the tokens per second that should be transferred to the recipient. Since it is possible to pick values that will transfer tokens but end up not transferring any tokens per second, the time at which an account can claim all of its tokens can be before the end of the vesting schedule.
Since the total amount of tokens that are vesting will stay constant, this does not allow anyone to get any tokens earlier by e.g. transferring tokens between multiple accounts. The result is just that the vesting schedule might not correct, related to and exacerbating the non-linear vesting schedule issue from above.

Exploit Scenario: - A user has 1000 tokens, the vesting takes place over 100 seconds, i.e. 10 tokens per second (tps)

- A now transfers 99 tokens to another party, this means `10 * 99 / 1000` tps will be transferred, which is 0
- So now A has 901 tokens but still 10 tps, which means all of their tokens will have vested after 91 seconds already. This step can be repeated.

Recommendation: Same recommendation as to the non-linear vesting schedule issue.

Update: According to the dev team "Requirements for PreJPEG changed and the token isn't transferrable anymore.". Fixed in this PR.

QSP-7 Share Dilution Concerns in LPFarming

Severity: *Low Risk*

Status: Fixed

File(s) affected: `farming/LPFarming.sol`

Description: Whenever a new pool is added using the `add` function, the `totalAllocPoint` will be increased, reducing the effective share of the existing pools by diluting them. This is only an issue if the function is called with `_withUpdate=false` since the pools will otherwise compute their rewards before the dilution takes place. The amount previous contributors lose through this dilution depends on when the the last update took place.

Recommendation: Either always update the pools before adding a new one or simulate realistically scenarios to find out how much previous contributors can realistically lose because of if it.

Update: Fixed in this PR.

QSP-8 Missing Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `vaults/ERC20VaultForDAO.sol`, `vaults/NFTVault.sol`

Description: The setter functions to control several contract settings does not check whether the supplied argument is within the allowed bounds. The documentation states that

> Only Dao can borrow 100% worth of Stablecoin based on the deposited amount of collateral asset

Currently, nothing prevents a malicious or accidental assignment of more than 100%.
List of functions:

- `ERC20VaultForDAO`
  - `setCreditLimitRate`
- `NFTVault`
  - `setCreditLimitRate`
  - `setInsurancePurchaseRate`
  - `setLiquidationLimitRate`

- . `setInsuranceLiquidationPenaltyRate`
- . `setOrganizationFeeRate`

Recommendation: Enforce the credit limit rate to be less or equal to 100%. Additionally, call the improved setter function from the constructor to also enforce this on deployment. Apply sensible sanity checks for the other values.

Update: Fixed in this PR.

QSP-9 Cannot Repay Less Than The Interest Owed

Severity: *Low Risk*

Status: Fixed

File(s) affected: `vaults/NFTVault.sol`

Description: The function `repay` calculates the amount that goes towards the principal payment by subtracting the interest by the amount that is currently being repaid.
`uint256 paidPrincipal = _amount - debtInterest;`
In case the amount is lower than the debt owed, the subtraction will underflow and the transaction will be reverted.

Recommendation: Check if the amount to be repaid is greater than the interest owed before performing the subtraction. In case it's not, set `paidPrincipal` to 0 and don't perform the subtraction.

Update: According to the dev team, it was "Fixed before the audit in the following PR: https://github.com/iceboxup/jpegd/pull/14".

QSP-10 Missing Validation In Credit Limit Rate

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ERC20VaultForDAO.sol`, `NFTVault.sol`

Description: The `function setCreditLimitRate()` does not validate the numerator and denominator of the struct `Rate`. Thus, the denominator can accept the value zero, and/or the numerator can be higher than the denominator. Although the function is only called by the owner, we recommend checking that the credit limit rate is within a reasonable interval. A similar issue was found for `NFTVault.setCreditLimitRate`.

```
function setCreditLimitRate(Rate memory _creditLimitRate)
    external
    onlyOwner
{
    creditLimitRate = _creditLimitRate;
}
```

Recommendation: Define reasonable intervals for the credit limit rate. Assure that the numerator is always less than the denominator. Assure that the denominator is always greater than zero. Consider imposing ranges when setting the numerator and denominator of the data structure `Rate`.

Update: Fixed in this PR.

QSP-11 Absence Of Events Logging

Severity: *Low Risk*

Status: Fixed

File(s) affected: `vaults/ERC20VaultForDAO.sol`

Description: The code is not triggering events, which will make the management and investigation of issues in runtime very difficult.

Recommendation: Trigger (log) events in every important action performed by the system.

Update: Fixed in this PR.

QSP-12 Safe Transfer Is Not Fully Compatible

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `CryptoPunksHelper.sol`

Description: According to **natspec** for CryptoPunksHelper in L36 on this contract `We aren't calling onERC721Received on the _to address because punks don't implement the ERC721 interface, but we are including this function for compatibility`. Is not clear if this refers to compatibility with ERC721 but in that case `onERC721Received()` must be called on address `_to` after checking that `_to` is a contract. If you don't implement properly ERC721 this will be confusing for other contracts composed with this one.

Recommendation: Check if `_to` is a contract and call `onERC721Received()`, otherwise document why you need this except if this is common practice in some NFT platforms.

Update: Fixed in this PR.

QSP-13 Ether Can Be Locked In With ERC20 Deposit

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ERC20VaultForDAO.sol`

Description: As function `ERC20VaultForDAO.deposit()` is `payable`, when the user deposits ERC20 it can also deposit ETH at the same time, this will lead to locked up ETH in the contract.

Recommendation: Check that `msg.value` is zero for `collateralAsset != ETH` or add a sweeping function so the owner can retrieve any locked ETH in the contract.

Update: Fixed in this PR.

QSP-14 Allowing Operations With Zero Amounts

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ERC20VaultForDAO.sol`

Description: In `ERC20VaultForDAO` several financial functions (`deposit()`, `borrow()`, `repay()`, `withdraw()`) allow valid operations with zero amounts, resulting in operations that are void of value for the contract and can be considered as some form of spurious, filler or wash-trading if executed.

Recommendation: Check that amounts are positive or above a minimum value to be considered valid.

Update: Fixed in this PR.

QSP-15 Minor `lockedTokens` View Calculation Error

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PreJPEGD.sol`

Description: In `_calculateClaimInRange()` function, that is only used in `view` function `vestingSchedule()` (not affecting contract state), the calculation return a rate in one case (`_startTimestamp >= _endTimestamp`) and an absolute number of token in the other case (`_startTimestamp < _endTimestamp`). The first case is incorrect because the function is used to view the number of locked tokens, not the rate.

Recommendation: Should return zero tokens in the first case, as the remaining locked tokens are zero as the end of the vesting period has been reached.

Update: According to the dev team "Requirements for PreJPEG changed and the token isn't transferrable anymore.". Fixed in this PR.

QSP-16 `rewardDebt` Variable In `LPFarming` Can Be Simplified

Severity: *Informational*

Status: Fixed

File(s) affected: `farming/LPFarming.sol`,

Description: The variable `rewardDebt` is hard to understand, especially due to the misleading name (there is no debt involved). The variable is only used to keep track of how many rewards have been claimed already.

Recommendation: Simplify the computation by replacing the `rewardDebt` variable with one named `lastAccRewardPerShare` that is updated each time the current `rewardDebt` is updated. It simply records the current value of `poolInfo.accRewardPerShare` at that time. The computation of `_withdrawReward` can then be simplified to the following, which is a lot easier to understand:

```
user.amount * (poolInfo[_pid].accRewardPerShare - user.lastAccRewardPerShare) / 1e36
```

Update: According to the dev team "Replaced rewardDebt with lastAccRewardPerShare". Fixed in this PR.

QSP-17 Complex Calculation Concerns In `_transfer`

Severity: *Informational*

Status: Fixed

File(s) affected: `vesting/PreJPEGD.sol`

Description: The way `transferredClaimed` is computed is hard to follow since it reuses a previous result in a non-obvious way.

```
// uses built-in arithmetic operators for clarity
uint256 transferredTokensPerSecond =
    sender.tokensPerSecond * _amount / fromBalance;
uint256 transferredClaimed =
    sender.claimedTokens * transferredTokensPerSecond / sender.tokensPerSecond;
```

When inlining the transferredTokensPerSecond used in the computation of `transferredClaimed`, this is what's left, which is a lot easier to understand.

```
uint256 transferredClaimed =
    sender.claimedTokens *  amount / fromBalance;
```

Recommendation: Simplify the computation to make it easier to follow.

Update: According to the dev team "Requirements for PreJPEG changed and the token isn't transferrable anymore.". Fixed in this PR.

QSP-18 SafeMath Not Needed For Solidity Versions >= 0.8

Severity: *Informational*

Status: Fixed

File(s) affected: `vesting/PreJPEGD.sol`

Description: Overflow detection is enabled by default starting with solidity version 0.8.
See https://docs.soliditylang.org/en/breaking/080-breaking-changes.html

Recommendation: Remove the SafeMath import and use built-in arithmetic operators for better readability.

Update: According to the dev team "Requirements for PreJPEG changed and the token isn't transferrable anymore.". Fixed in this PR.

QSP-19 Use `immutable` For Members Only Changed In Constructor

Severity: Informational

Status: Fixed

Description: Several contracts initialize their members in the constructor and don't provide any functions to change them. To make this intention explicit and get help from the compiler to not accidentally change them, use the `immutable` keyword.
Note: Only relevant for non-upgradeable contracts.

- `farming/LPFarming.sol`
  - `jpegd`
  - `rewardPerBlock`
- `vesting/PreJPEGD.sol`
  - `vestingStartTimestamp`
  - ` vestingEndTimestamp
  - `cliffEndTimestamp`
  - `token`

Recommendation: Use compiler support to enforce the `const` qualifier after contract construction.

Update: Fixed in this PR.

QSP-20 Unexplained Magic Numbers

Severity: Informational

Status: Fixed

Description: There are several instances where magic numbers are used in the code without providing context on how those were computed or what they are for. Some of them can be replaced by built-in solidity units and currently make maintenance and auditing more difficult.

- `vaults/ERC20VaultForDAO.sol`
  - L26, `SEC_YEAR`, use `365 days`
  - L50, use `1 ether`
  - L69, `return uint256(answer) * 10**10`, unclear why `10**10`
- `vaults/NFTVault.sol`
  - L26, `SEC_YEAR`, use `365 days`
  - L296, replace `10**18` with `ether`
  - L300 and L304, unclear why `10 ** 10`

Recommendation: If possible replace the magic numbers with solidity built-in types. Otherwise document how the values were computed.

Update: Fixed in this PR.

QSP-21 Inaccurate Name for ERC20VaultForDAO

Severity: Informational

Status: Fixed

File(s) affected: `vaults/ERC20VaultForDAO.sol`

Description: The name of the contract suggests only ERC20 tokens are stored in the vault. However, in the implementation, storing ETH directly is also supported.

Recommendation: Either change the name to something more general or remove ETH support.

Update: Fixed in this PR.

QSP-22 Code Duplication In `NFTVault.borrow`

Severity: Informational

Status: Fixed

File(s) affected: `vaults/NFTVault.sol`

Description: The following piece of code can be simplified by using a single `if` statement and reusing the code that both paths take.

```
// mint stablecoin
if (position.borrowType == BorrowType.USE_INSURANCE || _useInsurance) {
    uint256 feeAmount = ((_amount *
        settings.insurancePurchaseRate.numerator) /
        settings.insurancePurchaseRate.denominator) + organizationFee;
    // insurance & organization fee amount to dao
    totalFeeCollected += feeAmount;

    // remaining amount to user
    stablecoin.mint(msg.sender, _amount - feeAmount);
} else {
    // organization fee amount to dao
    totalFeeCollected += organizationFee;

    // remaining amount to user
    stablecoin.mint(msg.sender, _amount - organizationFee);
}
```

Recommendation: Replace with the following.

```
uint256 feeAmount = organizationFee;
if (position.borrowType == BorrowType.USE_INSURANCE || _useInsurance) {
    feeAmount += (_amount * settings.insurancePurchaseRate.numerator)
        / settings.insurancePurchaseRate.denominator;
}
totalFeeCollected += feeAmount;
stablecoin.mint(msg.sender, _amount - feeAmount);
```

Update: Fixed in this PR.

QSP-23 Gas Optimization In Mapping

Severity: *Informational*

Status: Fixed

File(s) affected: `vaults/NFTVault.sol`

Description: The `positions` mapping is never cleared when a position is closed. Instead, only the `positionOwner` entry is updated.

Recommendation: Clear the unused mapping entries after they are no longer needed to receive gas refunds.

Update: According to the dev team, it was "Fixed before the audit in the following PR: https://github.com/iceboxup/jpegd/pull/13 File: NFTVault.sol, Lines: 599, 646, 681, 705".

QSP-24 Renouncing The Ownership Is Possible

Severity: *Informational*

Status: Fixed

File(s) affected: `helpers/CryptoPunksHelper.sol`, `lock/JPEGDLock.sol`, `vaults/ERC20VaultForDAO.sol`, `farming/LPFarming.sol`, `vesting/PreJPEGD.sol`

Description: Many contracts inherit from some form of `Ownable`, which means the `renounceOwnership` function is available to call which would leave the contract in an unrecoverable state.

Recommendation: Consider overriding the `renounceOwnership` function and make it revert if this is your intention. Otherwise, document this important feature.

Update: Fixed in this PR.

QSP-25 Missing Docstrings And Inline Comments In The Code

Severity: *Informational*

Status: Fixed

Description: Almost the entire code does not present docstrings, which compromises the audit since the auditors do not have a clear description of what each snapshot of code is supposed to do.

Recommendation: Consider documenting each function of the code in terms of inputs, outputs, and processing, also adding inline comments in the code.

Update: Fixed in this PR.

QSP-26 Unlocked Pragma

Severity: *Informational*

Status: Acknowledged

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity ^0.8.0`. 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 notes from the dev team follows.

```
All the source files use the pragma string ^0.8.0, which locks the compiler version to version 0.8.x (above or equal to 0.8.0 but earlier than 0.9.0). From the solidity docs:

“The version pragma is used as follows: pragma solidity ^0.5.2;
A source file with the line above does not compile with a compiler earlier than version 0.5.2, and it also does not work on a compiler starting from version 0.6.0. (...)Because there will be no breaking changes until
version 0.6.0, you can be sure that your code compiles the way you intended. The exact version of the compiler is not fixed, so that bugfix releases are still possible.”
https://docs.soliditylang.org/en/v0.8.9/layout-of-source-files.html#version-pragma
Considering that the solidity compiler doesn’t introduce breaking changes in patch versions but only in minor and major versions, using the ^ syntax is ideal to benefit from compiler patches without any risk of breaking
changes.
```

QSP-27 Gas Optimization For Errors

Severity: *Informational*

Status: Acknowledged

Description: The cost of the `revert` operation is proportional to the string length passed as a parameter. Solidity 0.8 introduced another way of reporting errors, which incurs less gas usage. Basically, we define an error object, and such object is passed as a parameter to the `revert` command. Notice that such error objects can be defined outside of the contract, allowing the developer to create one single file defining all the possible errors. Errors also accept parameters. More information is available at: https://docs.soliditylang.org/en/v0.8.9/contracts.html#errors-and-the-revert-statement

Recommendation: In order to save gas, adopt the error scheme proposed in Solidity 0.8.x.

Update: No comments from the dev team.

QSP-28 Contract Size Optimizations

Severity: *Informational*

Status: Fixed

Description: Every time we declare a public variable, the Solidity compiler automatically generates the corresponding `get()` and `set()` functions. The contract size can also be optimized by turning `modifiers` into `functions`, as shown in the example below:

```
modifier checkStuff() {}
function doSomething() checkStuff {}
```

The `function doSomething()` can rewritten without `modifiers` as:

```
function checkStuff() private {}
function doSomething() { checkStuff(); }
```

Still focusing on optimizing the contract size, we must declare the correct visibility for functions and state variables. Functions or variables that are only called from the outside must be `external` instead of `public`. Functions or variables called only from within the contract must be `private` or `internal` instead of `public`. For instance, the `function min(uint256 a_, uint256 b_)` is clearly an `internal` function. Reference: https://ethereum.org/en/developers/tutorials/downsizing-contracts-to-fight-the-contract-size-limit/#taking-on-the-fight

Recommendation: Double-check the code and turn all possible `public` variables into `private` variables. Also, place the proper visibility of functions in order to optimize the contract size and reduce the deployment cost.

Update: Notes from the dev team follow.

```
We checked the code, we weren't able to find any function/variable with the wrong visibility. All the contracts are below the size limit.
```

QSP-29 Time Period Can Be More Accurate

Severity: *Informational*

Status: Acknowledged

File(s) affected: `ERC20VaultForDAO.sol`

Description: Using seconds per year used for interest rates calculation as `86400 * 365` can be more accurate if we consider leap years, one day extra every 4 years. Then seconds per year will be `86400 * 365.25` or in integer terms will be `86400 * 1461 / 4`.

Recommendation: Consier using `86400 * 1461 / 4` as `SECS_YEAR`.

Update: Notes from the dev team follow.

```
We switched from using SECS_YEAR to using 365 days, considering leap years isn't really needed in our case and we decided to keep the code as is for better readability.
```

QSP-30 Convoluted Unclaimed Transfer Semantics

Severity: *Undetermined*

Status: Fixed

File(s) affected: `PreJPEGD.sol`

Description: The transfer of `PreJPEGD` has a semantic that can difficult be to understand. For example, if Alice has claimed 50% of its PreJPGD tokens and transfers 50% of its PreJPGD balance to Bob, then Bob receives 1/2 of claimed tokens and 1/2 of unclaimed tokens. Is possible, that a more natural semantics will be that Alice can only transfer unclaimed tokens, and never transfers claimed tokens.

Recommendation: Consider if `PreJPEGD._transfer()` should only allow the transference of unclaimed tokens, otherwise improve the documentation of the special semantics of this function.

Update: Notes from the dev team follow.

```
Requirements for PreJPEG changed and the token isn't transferrable anymore. https://github.com/iceboxup/jpegd/pull/39
```

Automated Analyses

Slither

Slither has detected many results out of which the majority have been filtered out as false positives and the rest have been integrated into the findings from this report.

Code Documentation

1. Regarding the next line in specs:

    The DAO will initially seed liquidity for PUSd among a basket of other tokens in order to peg its value as close to $1 as possible at all times

So will there be multiple ERC20VaultForDAO contracts? Because currently only a a single type of collateral is supported. Also, what happens if the price of the underlying collateral changes? E.g. the value of eth goes up, won't the value of PUSd go up as well?

1. Related to that

    DAO can also mint PUSD using USDC 1:1

And according to the diagram, NFTVault will be able to mint PUSd as well when users borrow against their NFTs. Overall I don't see how the price will be stable.

1. Regarding the following line:

    Only Dao can borrow 100% worth of Stablecoin based on the deposited amount of collateral asset.

Currently only up to `collateral * creditLimitRate` can be borrowed (once the borrow function is fixed to actually enforce that).

Adherence to Best Practices

1. Tests should be using typescript to make interaction with the contracts easier and to catch changes in the API during active development. Bindings are already created with the `typechain` hardhat plugin. **Fixed**

2. There are very few comments documenting how the code should work. Ideally there should be some high-level comment on the contract level to explain the big picture of what the contract is trying to achieve in addition to inline comments that register assumptions or non-obvious functionality. **Fixed**

3. Tests can have deeper effects if values in one test are adjusted it can break other tests. This makes it very hard to adjust the tests without having to change all of the others.
   E.g. `ERC20VaultForDao.js` uses a global `usdcVault` and expects specific balances after some of the tests. **Fixed: many tests added**

4. Use Unicode strings directly instead of encoding them. Since solidity version 0.7 it is possible to use `unicode"JPEG'd"` instead of `"JPEG\xE2\x80\x99d"` **Fixed**

5. Consider using interface types `IInterface` for members directly instead of `address` and casting it to the interface on each usage.
   `ERC20VaultForDao`: `stablecoin`, `oracle` **Fixed**

6. Function that is part of the public API but not used by the contract should be made `external` instead of `public`. See the output of `slither .` and look for "should be declared external". **Fixed**

7. Use the checks-effects-interacts pattern when external calls are involved.
   It is generally good practice to guard against reentrancy this way even if the functions have a `nonReentrant` modifier applied to them or the external call is to a contract under one's control. Code might get copied and pasted and adjusted for a different use case where this assumption might no longer hold, leading to potential issues later.
   Additionally, it can make using static analyzers such as `slither` more effective by removing false positives. **Fixed**

8. There are still some placeholder names or the comment is old on L11 from `IOracle.sol`, remove the comment of use the actual names. **Fixed**

Test Results

Test Suite Results

No test failed from a total of 74 tests. The most critical contract, `NFTVault`, has a total of 24 passing tests. The tests suite is currently considered Medium quality but will be High quality if more tests are included considering the reported issues.

Update: on reaudit the number of tests climbed to 119 test cases. We consider the testing suite very complete and extensive. Detailed output is included below.

```
$ yarn test
yarn run v1.22.10
$ hardhat test
No need to generate any newer typings.

  Controller
     ✓ should return the correct JPEG balance (66ms)
     ✓ should allow the vault to withdraw jpeg (119ms)
     ✓ should allow admins to set the fee address (21ms)
     ✓ should allow strategists to set vaults for tokens (27ms)
     ✓ should allow admins to approve and revoke strategies (52ms)
     ✓ should allow strategists to set strategies (78ms)
     ✓ should deposit tokens into the strategy when calling earn (53ms)
     ✓ should allow strategists to withdraw all tokens from a strategy (63ms)
     ✓ should allow strategists to withdraw tokens (26ms)
     ✓ should allow strategists to withdraw tokens from a strategy (51ms)
     ✓ should allow vaults to withdraw tokens from a strategy (69ms)

  CryptoPunksHelper
     ✓ should not allow the owner to renounce ownership (7ms)
     ✓ should return the owner of this contract when the nft is owned by the helper (27ms)
     ✓ should return the nft owner in all other cases (18ms)
     ✓ should only allow the owner to call transferFrom and safeTransferFrom (25ms)
     ✓ should revert if neither the contract or the precompute address hold the nft (779ms)
     ✓ should keep the nft if the recipient is the owner (446ms)
     ✓ should send the nft if the recipient is anyone besides the owner (442ms)
     ✓ should allow the owner to send nfts (48ms)
     ✓ can't precompute with the 0 address or the contract as the owner (10ms)

  FungibleAssetVaultForDAO
     ✓ should be able to update creditLimitRate (126ms)
     ✓ should be able to deposit assets (167ms)
     ✓ should be able to borrow assets (155ms)
     ✓ should be able to repay assets (217ms)
     ✓ should be able to withdraw assets (137ms)

  JPEG
     ✓ should allow the minter to mint tokens (18ms)
     ✓ shouldn't allow users to mint tokens (15ms)

  JPEGLock
     ✓ should not allow the owner to renounce ownership (5ms)
     ✓ Only owner can lock tokens (46ms)
     ✓ Cannot unlock before 1 year (74ms)
     ✓ Only position owner can unlock after 1 year (67ms)

  JPEGStaking
     ✓ stake should not work with invalid parameters (21ms)
     ✓ stake should work (36ms)
     ✓ unstake should not work with invalid parameters (12ms)
     ✓ unstake should work (69ms)

  LPFarming
     ✓ should not allow the owner to renounce ownership (4ms)
     ✓ only owner can add pools (62ms)
     ✓ only owner can update pool configuration (61ms)
     ✓ should not allow an epoch with invalid parameters (28ms)
     ✓ should update epoch (88ms)
     ✓ should not emit tokens outside of an epoch (115ms)
     ✓ should not assing rewards in between epochs (113ms)
     ✓ should not allow non whitelisted contracts to farm (52ms)
     ✓ should not allow 0 token deposits or withdrawals (28ms)
     ✓ should work for zero allocations (58ms)
     ✓ should allow whitelisted contracts to farm (85ms)
1:
owner reward:  50000
alice reward:  0
bob reward:  0
2:
owner reward:  16716
alice reward:  33333
bob reward:  0
3:
owner reward:  41762
alice reward:  50053
bob reward:  8333
4:
owner reward:  71787
alice reward:  60069
bob reward:  18341
5:
owner reward:  96847
alice reward:  8343
bob reward:  35027
6:
owner reward:  126872
alice reward:  8351
bob reward:  55044
7:
owner reward:  139432
alice reward:  8351
bob reward:  37519
     ✓ users can deposit/withdraw/claim (8922ms)

  NFTVault
```

```
    ✓ should be able to borrow (229ms)
    ✓ should be able to borrow with insurance (112ms)
    ✓ should be able to repay (388ms)
    ✓ should be able to close position (285ms)
    ✓ should be able to liquidate borrow position without insurance (583ms)
    ✓ should be able to liquidate borrow position with insurance (291ms)
    ✓ should be able to repurchase (296ms)
    ✓ should allow the liquidator to claim an nft with expired insurance (304ms)
    ✓ get ape punk + open position + borrow 600ETH (102ms)
    ✓ get punk + increase debt limit to 50000ETH + open position + borrow 6000ETH (212ms)
    ✓ organization is deducted from debt (62ms)
    ✓ insurance fee is deducted from debt (67ms)
    ✓ collect mints interest and send to dao (265ms)
    ✓ should allow the dao to override floor price (128ms)
    ✓ should allow the dao to set nftType (86ms)
    ✓ should allow the dao to set the value of an nft type (64ms)
    ✓ should be able to update borrowAmountCap (52ms)
    ✓ should be able to update debtInterestApr (61ms)
    ✓ should be able to update creditLimitRate (104ms)
    ✓ should be able to update liquidationLimitRate (100ms)
    ✓ should be able to update organizationFeeRate (59ms)
    ✓ should be able to update insurancePurchaseRate (63ms)
    ✓ should be able to update insuranceLiquidationPenaltyRate (72ms)

  PreJPEG
    ✓ should mint PreJPEG tokens on new vesting (35ms)
    ✓ should burn all tokens on revoke (50ms)
    ✓ should burn tokens on release (64ms)
    ✓ should not allow transfers (37ms)

  Stablecoin
    ✓ MINTER can mint tokens (39ms)
    ✓ PAUSER can pause token transfer (68ms)
    ✓ PAUSER can unpause token transfer (82ms)

  StrategyPUSDConvex
    ✓ should return the correct name (5ms)
    ✓ should return the correct JPEG balance (44ms)
    ✓ should allow the DAO to change controller (20ms)
    ✓ should allow the DAO to change usdc vault (18ms)
    ✓ should deposit want on convex (35ms)
    ✓ should allow the controller to withdraw JPEG (2987ms)
    ✓ should allow the controller to withdraw non strategy tokens (1261ms)
    ✓ should allow the controller to withdraw want (192ms)
    ✓ should allow the controller to call withdrawAll (120ms)
    ✓ should add liquidity with pusd when harvest is called and curve has less pusd than usdc (18789ms)
    ✓ should add liquidity with usdc when harvest is called and curve has less usdc than pusd (742ms)
    ✓ should revert on deploy with bad arguments (603ms)

  TokenSale
    ✓ should return the correct tokens when calling getSupportedTokens (3ms)
    ✓ should return the correct oracles when calling getTokenOracles (4ms)
    ✓ should return the correct oracle when calling getOracle (4ms)
    ✓ should allow the owner to allocate tokens (42ms)
    ✓ should allow the owner to set the sale schedule (68ms)
    ✓ should allow users to deposit (331ms)
    ✓ should allow the owner to finalize the raise (205ms)
    ✓ should allow the owner to enable withdrawals (94ms)
    ✓ should allow users to withdraw (265ms)
    ✓ should allow the owner to transfer the raise to treasury (257ms)

  TokenVesting
    ✓ should allow members of the vesting_controller role to vest tokens (76ms)
    ✓ shouldn't allow to release vested tokens before vesting starts (30ms)
    ✓ should allow users to release (73ms)
    ✓ should not emit tokens during the cliff period (35ms)
    ✓ should allow to claim all the tokens after vesting is over (62ms)
    ✓ should allow the owner to revoke tokens (120ms)

  yVault
    ✓ should have the same decimals as the deposit token (22ms)
    ✓ should return the correct JPEG balance (15ms)
    ✓ should allow the farm to withdraw JPEG (57ms)
    ✓ should allow users to deposit (93ms)
    ✓ should mint the correct amount of tokens (106ms)
    ✓ should deposits tokens into the strategy when calling earn (77ms)
    ✓ should withdraw the correct amount of tokens (184ms)
    ✓ should not allow non whitelisted contracts to deposit and withdraw (24ms)
    ✓ should allow whitelisted contracts to deposit/withdraw (82ms)

  yVaultLPFarming
    ✓ should allow users to deposit tokens (74ms)
    ✓ should allow users to withdraw (101ms)
    ✓ should allow users to claim (293ms)
    ✓ should not allow non whitelisted contracts to farm (68ms)
    ✓ should allow whitelisted contracts to farm (88ms)
```

Solc version: 0.8.0		Optimizer enabled: true		Runs: 1000	Block limit: 30000000 gas	
Methods						
Contract	Method	Min	Max	Avg	# calls	usd (avg)
Controller	approveStrategy	47274	47286	47285	34	-
Controller	earn	-	-	60063	1	-
Controller	grantRole	-	-	51735	37	-
Controller	inCaseStrategyTokensGetStuck	61329	62283	61806	2	-
Controller	inCaseTokensGetStuck	-	-	57198	1	-
Controller	revokeStrategy	-	-	25413	1	-
Controller	setFeeAddress	-	-	26623	1	-
Controller	setStrategy	49694	90709	50911	34	-
Controller	setVault	-	-	47374	22	-
Controller	withdraw	-	-	65402	1	-
Controller	withdrawAll	69055	88956	79006	2	-
CryptoPunks	getPunk	-	-	73143	6	-
CryptoPunks	transferPunk	-	-	65051	4	-
ERC20PresetMinterPauser	grantRole	58688	59102	59004	120	-
ERC20PresetMinterPauser	revokeRole	34783	35199	34982	48	-
ERC20Upgradeable	approve	-	-	59963	2	-
ERC20Upgradeable	transfer	51268	65613	57937	10	-
ERC20Upgradeable	transferFrom	131290	140678	135984	2	-
ERC20Vault	borrow	99346	133546	117998	11	-
ERC20Vault	deposit	57348	102545	91522	11	-
ERC20Vault	repay	59168	63866	62300	3	-
ERC20Vault	setCreditLimitRate	37105	39976	38062	3	-
ERC20Vault	setDebtInterestApr	-	-	37184	1	-
ERC20Vault	setLiquidationLimitRate	-	-	39994	1	-
ERC20Vault	withdraw	63933	99261	81597	2	-
ERC721	safeTransferFrom	-	-	84572	1	-
JPEG	approve	46783	46819	46804	35	-
JPEG	grantRole	-	-	51807	39	-
JPEG	mint	72822	107022	99247	11	-
JPEG	transfer	52024	56800	56000	6	-
JPEGLock	lockFor	-	-	133714	3	-
JPEGLock	transferOwnership	-	-	28791	23	-
JPEGLock	unlock	-	-	67020	2	-
JPEGStaking	stake	-	-	170847	2	-
JPEGStaking	unstake	-	-	107339	1	-
LPFarming	add	82477	142070	118298	13	-

LPFarming	claim	107516	143716	125616	2	-
LPFarming	claimAll	138298	155398	146848	2	-
LPFarming	deposit	99135	126476	112668	10	-
LPFarming	newEpoch	42639	147178	110751	8	-
LPFarming	set	36322	44411	41715	3	-
LPFarming	setContractWhitelisted	-	-	46644	1	-
LPFarming	withdraw	52460	106937	85044	4	-
MockV3Aggregator	updateAnswer	78889	120737	102681	8	-
NFTVault	borrow	144550	429919	403893	15	-
NFTVault	claimExpiredInsuranceNFT	-	-	104676	1	-
NFTVault	closePosition	-	-	91412	1	-
NFTVault	collect	76009	87411	83610	3	-
NFTVault	disableFloorOverride	-	-	36042	1	-
NFTVault	finalizePendingNFTValueETH	-	-	227849	1	-
NFTVault	liquidate	169865	189478	184575	4	-
NFTVault	overrideFloor	-	-	58673	1	-
NFTVault	repay	91862	108686	99699	4	-
NFTVault	repurchase	-	-	129909	1	-
NFTVault	setBorrowAmountCap	-	-	36478	2	-
NFTVault	setInsuranceLiquidationPenaltyRate	-	-	37249	1	-
NFTVault	setInsurancePurchaseRate	-	-	37227	1	-
NFTVault	setNFTType	-	-	64964	1	-
NFTVault	setNFTTypeValueETH	-	-	37147	1	-
NFTVault	setOrganizationFeeRate	-	-	37242	1	-
NFTVault	setPendingNFTValueETH	-	-	62529	1	-
PreJPEG	grantRole	-	-	51785	4	-
PreJPEG	release	-	-	148585	1	-
PreJPEG	revoke	-	-	110845	1	-
PreJPEG	vestTokens	-	-	256003	4	-
StableCoin	approve	46807	47095	46857	13	-
StableCoin	grantRole	101252	118736	109410	89	-
StableCoin	mint	-	-	73544	3	-
StableCoin	pause	-	-	47185	2	-
StableCoin	revokeRole	-	-	46965	24	-
StableCoin	transfer	32492	49592	41254	10	-
StableCoin	unpause	-	-	25231	1	-
StrategyPUSDConvex	deposit	-	-	85978	1	-
StrategyPUSDConvex	grantRole	-	-	51743	12	-
StrategyPUSDConvex	harvest	635854	747869	691862	2	-
StrategyPUSDConvex	setController	-	-	29385	1	-
StrategyPUSDConvex	setUSDCVault	-	-	29386	1	-
TestERC20	approve	46747	46807	46792	34	-
TestERC20	mint	56456	73568	72623	73	-
TestERC20	setDecimals	-	-	26706	12	-
TestERC20	transfer	54378	54414	54404	5	-
TestERC721	approve	48990	49014	49011	14	-
TestERC721	mint	51930	69030	66461	20	-
TokenSale	allocateTokensForSale	-	-	90012	7	-
TokenSale	deposit	133365	135611	134488	8	-
TokenSale	depositETH	67137	155537	84817	5	-
TokenSale	enableWithdrawals	-	-	48662	3	-
TokenSale	finalizeRaise	-	-	131273	4	-
TokenSale	setSaleSchedule	70599	70611	70609	6	-
TokenSale	transferToTreasury	-	-	101861	1	-
TokenSale	withdraw	78795	80891	79843	4	-
TokenVesting	grantRole	-	-	51712	6	-
TokenVesting	release	91748	97005	94377	4	-
TokenVesting	revoke	-	-	74599	1	-
TokenVesting	vestTokens	139456	168992	152623	7	-
WETH	approve	46727	46739	46733	4	-
WETH	deposit	-	-	67962	4	-
YVault	approve	-	-	46774	6	-
YVault	deposit	73973	134896	119511	4	-
YVault	depositAll	75224	136147	119619	11	-
YVault	earn	81967	143030	122676	3	-
YVault	setContractWhitelisted	-	-	46677	1	-
YVault	setFarmingPool	-	-	46306	19	-
YVault	transfer	-	-	47378	2	-
YVault	withdraw	72492	116159	94348	3	-
YVault	withdrawAll	82058	120272	96449	3	-
YVault	withdrawJPEG	74897	161247	103680	3	-
YVaultLPFarming	claim	108258	167348	137803	2	-
YVaultLPFarming	deposit	131030	167367	138737	5	-
YVaultLPFarming	setContractWhitelisted	-	-	46610	1	-
YVaultLPFarming	withdraw	-	-	95191	1	-

Deployments			% of limit	
Controller	1690923	1690935	1690934	5.6 %	-
CryptoPunks	-	-	1951136	6.5 %	-
CryptoPunksHelper	-	-	1172380	3.9 %	-
FungibleAssetVaultForDAO	-	-	1966477	6.6 %	-
JPEG	2681099	2721007	2706671	9 %	-
JPEGLock	872481	872493	872492	2.9 %	-
JPEGStaking	-	-	2672449	8.9 %	-

```
|  LPFarming                                                      ·          -  ·          -  ·     1810376  ·          6 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  MockBooster                                                    ·          -  ·          -  ·      274014  ·        0.9 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  MockCurve                                                      ·          -  ·          -  ·      574348  ·        1.9 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  MockRewardPool                                                 ·     534810  ·     623360  ·      544477  ·        1.8 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  MockStrategy                                                   ·     566473  ·     566485  ·      566482  ·        1.9 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  MockV3Aggregator                                               ·     537836  ·     537884  ·      537848  ·        1.8 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  NFTVault                                                       ·          -  ·          -  ·     4887472  ·       16.3 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  PreJPEG                                                        ·    3337959  ·    3337971  ·     3337968  ·       11.1 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  StableCoin                                                     ·          -  ·          -  ·     1991118  ·        6.6 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  StrategyPUSDConvex                                             ·          -  ·          -  ·     3513227  ·       11.7 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  TestERC20                                                      ·    2229578  ·    2229734  ·     2229652  ·        7.4 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  TestERC721                                                     ·          -  ·          -  ·     1467430  ·        4.9 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  TokenSale                                                      ·    2152437  ·    2152461  ·     2152459  ·        7.2 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  TokenVesting                                                   ·    1452520  ·    1452532  ·     1452531  ·        4.8 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  WETH                                                           ·          -  ·          -  ·      994443  ·        3.3 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  YVault                                                         ·    2431440  ·    2431452  ·     2431451  ·        8.1 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·
|  YVaultLPFarming                                                ·          -  ·          -  ·     1250370  ·        4.2 %  ·          -  |
·-----------------------------------------------------------------|------------|------------|------------|-------------|-------------·

  119 passing (6m)

Failed to generate 3 stack traces. Run Hardhat with --verbose to learn more.
 ✓  Done in 338.67s.
```

Code Coverage

The code coverage is overall very good, statement coverage is above 99%, branch coverage is above 95%, function coverage is above 98% and line coverage is above 99% also.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
escrow/	100	100	100	100	
NFTEscrow.sol	100	100	100	100	
farming/	99.32	98.39	100	100	
LPFarming.sol	100	100	100	100	
yVaultLPFarming.sol	98	95.83	100	100	
helpers/	100	83.33	100	100	
CryptoPunksHelper.sol	100	83.33	100	100	
lock/	100	100	100	100	
JPEGLock.sol	100	100	100	100	
sale/	100	100	100	100	
TokenSale.sol	100	100	100	100	
staking/	100	100	100	100	
JPEGStaking.sol	100	100	100	100	
tokens/	100	100	100	100	
JPEG.sol	100	100	100	100	
StableCoin.sol	100	100	100	100	
vaults/	98.82	88.89	98	99.22	
FungibleAssetVaultForDAO.sol	100	84.62	100	100	
NFTVault.sol	98.56	90.24	97.56	99.05	219,220
vaults/yVault/	100	100	100	100	
Controller.sol	100	100	100	100	
yVault.sol	100	100	100	100	
vaults/yVault/strategies/	100	98.44	100	100	
StrategyPUSDConvex.sol	100	98.44	100	100	
vesting/	100	100	100	100	
PreJPEG.sol	100	100	100	100	
TokenVesting.sol	100	100	100	100	
All files	**99.48**	**96.07**	**99.44**	**99.74**	

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

```
fbb2718b3f0ea4310d6fb4c2ecc525cefe78b7a59b78c9b70d9f0845ef7d948c ./contracts/interfaces/IJPEGLock.sol
b2d9404551b948a257acbcb6fe5d920f3b984f7922c5c69e2812d2ad8f6438e2 ./contracts/interfaces/IFloorOracle.sol
b3a4176580b6b0a3f92b7a4127b9795f06c0c5d2dbc0fc95ff947158430b0480 ./contracts/interfaces/IBooster.sol
bb22662301e43feff8a7846e2926c9fa2d9496ad2dfc83c07ab46a6df13158e0 ./contracts/interfaces/IStableCoin.sol
d2a22c131b871eef31e34b2bcba19894402c7f75de8bbaec06ede7a1febd51ed ./contracts/interfaces/IERC20Decimals.sol
c68bc30dca52adff3a1a8b66942a90610793cbef7903784757407a9cbac6ef32 ./contracts/interfaces/IUniswapV2Router.sol
e5696177afe38a2b170caeb46f0f29a23ca38ece0bbf492ce4571c4f5be0560e ./contracts/interfaces/IFungibleAssetVaultForDAO.sol
dbc7d0fc0bc779ece11ba0d3cc155eaab7ab2f9dd1f221e522206384d8813795 ./contracts/interfaces/IWETH.sol
763be192b9b52acaca11d084d7882ce70af329da92a8872f705dc8891bdedd25 ./contracts/interfaces/IStrategy.sol
ca4b005e880a674761f9ba6108c528598563f727f2b06120d650e9feebc073e5 ./contracts/interfaces/IYVault.sol
da7e5fa14107b8963d04d847015e920995cb0975d8d221becdce612d94cd45b1 ./contracts/interfaces/ISwapRouter.sol
343ba94f8b7203b9fc0e501eac7596a2d8ba7bcfe7f58194196df4d0c66f1f4c ./contracts/interfaces/IUniswapV2Factory.sol
b2ed849e166b7893f67d689f914f7b1cc78ced18baa6c4a04226f9f294462815 ./contracts/interfaces/IBaseRewardPool.sol
29624cf93e464ff66b1d13f2cba705244ce6ecfaaf850fd79628f2c145ddacee ./contracts/interfaces/ICryptoPunks.sol
c9bc8b0e46f5ff397ad2caa7ae3bba44874cccb03560cc8ba91ffc756e21069e ./contracts/interfaces/IUniswapV2Pair.sol
52e5e56ad8cf374d8b267c6831eb22ce593e7e83f684d86be605164b7ad1edd8 ./contracts/interfaces/ICurve.sol
afc8ba365941fab9659cdf48ceb5d16b7e9907b894af20feb88ce9da77c46f10 ./contracts/interfaces/IAggregatorV3Interface.sol
c5cecf95be2c76cb755fb98b931ba7a3e7fff0797755a4bfb77a24af3b8afc6a ./contracts/interfaces/IController.sol
21636dcee5517877362283ab72cd3e34d292092b2c572b205ffe2022d5cba38d ./contracts/helpers/CryptoPunksHelper.sol
af99b5e2fa39e8a81ba1ffadb7d2178b3969c115faaf7563fe3ba81d10a8f956 ./contracts/tokens/JPEG.sol
e57690b38301a3e5d7ad238f95cc2f78d2c567171e49d5b20f96655e057f89e4 ./contracts/tokens/StableCoin.sol
62ba60f100c3155e8f04d7f4aa3f3f17d4db33221ab20871da47d27eeaba6269 ./contracts/escrow/NFTEscrow.sol
b740ce6b813dfdb9fd4c227069bd944f002c19953339fd493b376094fbafc6c2 ./contracts/farming/yVaultLPFarming.sol
a4f3c95580176a8fd63e67af541102cbd08a9b24cc695640523d88fa39de0f0b ./contracts/farming/LPFarming.sol
fc3849fe8a3481cc31df18145857f941fc35ef226d4f1bb668df7c5047ec6212 ./contracts/staking/JPEGStaking.sol
199f5c79062740db79beb57cf5213b90097729c20f45958cd3bb541e3c2b4f00 ./contracts/vaults/FungibleAssetVaultForDAO.sol
009aa9e5e42d0d77a8116afff6982273f960a77e4a1216336d22dbf887670761 ./contracts/vaults/NFTVault.sol
a9e1572d3dd0b07172bc38e8f82a4aea672bf5ab447b6b7f2ba137c6c50474b8 ./contracts/vaults/yVault/Controller.sol
684b6ff53d887bcf2f4d00240354d7cb8e6411339b1fc16608d0b14e31e1ed15 ./contracts/vaults/yVault/yVault.sol
3cf97e6fa155c149efe43c8c5e9cc9480fee8305026d329dc13ef52303ba91fc ./contracts/vaults/yVault/strategies/StrategyPUSDConvex.sol
da560de20aacdb6fd7a9a6c66f9a70bf1edb94eb86cefa9975b6660969e82966 ./contracts/vesting/PreJPEG.sol
22fad40068bee6bec193f3d040f6bf7e393d98925c6bedc41afb38ce9038f30e ./contracts/vesting/TokenVesting.sol
606120fff1630ac389a339ed3b3e456fe2a73569f5149b5f17e7faa5be831308 ./contracts/lock/JPEGLock.sol
9ea89ef750068ec4c19e76d83b47741a12351ea8780d7a031fe4186de835277e ./contracts/test/TestERC721.sol
febfd3edd1957e0589153bce45350f73309e32ba95e0a00ee3d79d509135dff8 ./contracts/test/MockStrategy.sol
ad4ef26ea8ccc3565cf8e28ee513bd1795a3ba04cb5352d8496a0d90d492b343 ./contracts/test/MockBooster.sol
a630d8e8b8db8185865c48a139002bf325243f68bac9f3b74703f0611267e75f ./contracts/test/MockRewardPool.sol
2baa6018e4f4c9a00154018e9328fe984e6353ac6c1392f918bf6f12e7de8c83 ./contracts/test/CryptoPunks.sol
23b3807f1f615a00c70ac8755eb6ef59cc511e3479140eaec80f67b493cad8da ./contracts/test/WETH.sol
10ab190e79a41835b4520bc5e55aadd071b9c77779b34afc301817ac3a5b8763 ./contracts/test/TestERC20.sol
3012cf13bdc586afd6a57ad4c9ff09851e5af19431306709d5533b302382c078 ./contracts/test/MockCurve.sol
50df49831c248de57ecdadec04132a2065eda94a78230b2d361e4440a1c3e3ca ./contracts/test/MockAggregator.sol
cf57574396ab1d1f392c381401fc7e2ea59d2ea17b2e98f1069f3c32cf489414 ./contracts/draft/ERC20Vault.sol
f29adb1273d5049a0c1da7c76c720ec2e753a4613c6627cd3ec06037934d7406 ./contracts/sale/TokenSale.sol
```

Tests

```
916af868c2225c4623f29c2c00cb7c1c1b98d805df998ae848a3d2fad2f9d136 ./tests/PreJPEG.ts
cdf0bbc9c3378a1568971bc30ee9002b930f2f3d49d832e575c17214b9d74941 ./tests/CryptoPunksHelper.ts
5abfd17c454574b7d7f744f1b24e69965fba5a3b7bff5b8bad57d3544a9e960f ./tests/LPFarming.ts
0e3a7f8067b68ceba1928edd14b502b22fcb03058dd4a290b660abfd665c996c ./tests/NFTVault.ts
76513e470ac41ac2ea3df2f03461f1c6d352bad0c059938ba4e54714d2f9e3b8 ./tests/TokenVesting.ts
7570124fe11d57fc4de7710b528e766431964989d65238c88a5c180f74086dae ./tests/FungibleAssetVaultForDAO.ts
ede04591360eb8dba183e1e607ecffc454ea6430ee146c60e5ae0c72d24f9caa ./tests/Stablecoin.ts
0a3f234f29d76d75d966ab9451bb869ea043da1ed42584c3fece62115bac34ed ./tests/yVault.ts
7596c95d0145674bc614fcdb507a83cb79124eb7e91f52772f1013ab12f1982c ./tests/TokenSale.ts
70abc7f0792d7adae0b854779a03b18c042b87d9b6354a06754111773a4e6e02 ./tests/JPEGLock.ts
```

```
465efe6352d8df16d16e12a9b9874ac90cb7ecc7b0d34f969e6727a595b09e5a ./tests/utils.ts
4a53f5c2a7b6093b1c0fd22276000085a83ffa3f97a6921e6102795cf2b0632b ./tests/JPEG.ts
7d277ec9768ba79be7bcf65b0b12c0d40392015629ac235603971a572a546af5 ./tests/StrategyPUSDConvex.ts
bd0ee29d6273f8b09d4e28a43fdd8b96eacb01868357e8cce69faee16049efc1 ./tests/yVaultLPFarming.ts
2e57a065d04704a562ba2825d3ac8c890926565c0a13e2b00f4b5551dd6b536b ./tests/JPEGStaking.ts
5f2d066235667b5e9dc4a5d3e5f7516d166f37bd96d23cf5e7112709ff4f216b ./tests/Controller.ts
```

Changelog

- 2021-11-12 - Initial report
- 2021-12-02 - Reaudit 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.