Quantstamp Security Assessment Certificate

April 22nd 2022 — Quantstamp Verified

MetaStreet NFT Collateral Vault

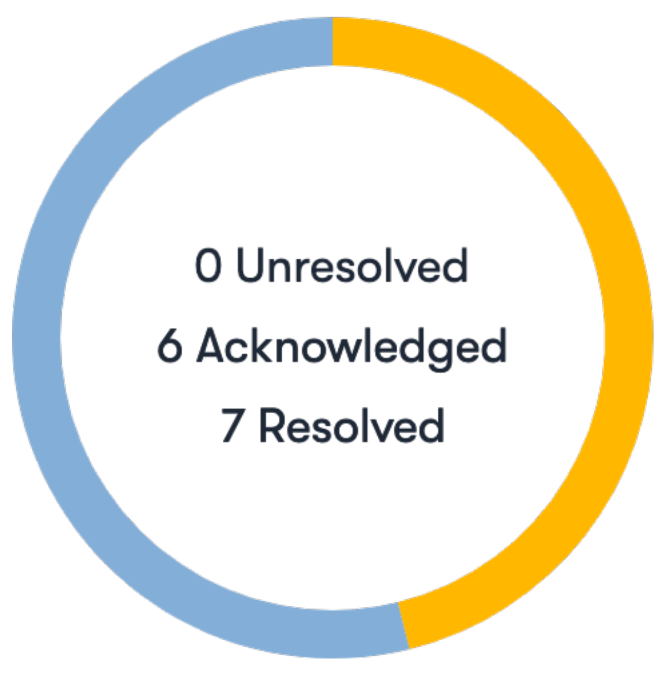
This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Lending platform
Auditors	Guillermo Escobero, Security Auditor Danny Aksenov, Security Auditor Alex Murashkin, Senior Software Engineer
Timeline	2022-04-12 through 2022-04-29
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	MetaStreet NFT Collateral Vault v1.0 - Technical Whitepaper (Draft 2) - Private
Documentation Quality	High
Test Quality	High
Source Code	

Repository	Commit
metastreet-contracts	a09c65f
metastreet-contracts (fixes)	29cfed7

Total Issues	**13** (7 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**4** (3 Resolved)
Low Risk Issues	**2** (1 Resolved)
Informational Risk Issues	**5** (1 Resolved)
Undetermined Risk Issues	**2** (2 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

Quantstamp audit team has reviewed the MetaStreet NFT Collateral Vault project, and drawn the following conclusions: it implements high-quality code with clear design and architecture, is well documented, and with high adherence to the specification provided. MetaStreet team provided a whitepaper and a video walkthrough describing the architecture and related use cases. Several issues were found and included in this report.

After re-audit: The MetaStreet team addressed all the issues found and provided clarifications of the fixes done. These explanations are included in this report. After discussion with the MetaStreet team, we decided to classify QSP-10 as a false positive. *Adherence to Specification* suggestions are not confirmed to be resolved as Quantstamp did not receive a new draft for the whitepaper. *Best Practices* suggestions were resolved.

ID	Description	Severity	Status
QSP-1	Privileged Roles and Ownership	^ Medium	Acknowledged
QSP-2	Lack Of Validation On The Loans Sold To The Vault	^ Medium	Fixed
QSP-3	Oracle Centralization	^ Medium	Mitigated
QSP-4	Concerns On Admin Fee System	^ Medium	Fixed
QSP-5	Serviced Loans Are Not Removed From Pending Loans	⌄ Low	Acknowledged
QSP-6	Missing Input Validation	⌄ Low	Fixed
QSP-7	Interaction with External Contracts	○ Informational	Acknowledged
QSP-8	Race Conditions / Front-Running	○ Informational	Acknowledged
QSP-9	Deprecated Function	○ Informational	Fixed
QSP-10	[Not An Issue] Concerns On Upgradability	○ Informational	Acknowledged
QSP-11	Presence Of Unused Variables	○ Informational	Acknowledged
QSP-12	Missing Address Validation	? Undetermined	Fixed
QSP-13	Zero-Value Transfers	? 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.

DISCLAIMER: This report only covers the files scoped in the audit process. Please note that the implementation of the *INoteAdapter* interface was not provided and, therefore, not included in this audit. Because of that, we were not able to confirm its correct operation and interaction with the rest of the contracts. Please refer to the appendix section "File Signatures" for the list of audited files and their digest values. Update: commit 29cfed7 included new contract files: VaultRegistry.sol and three more in "./contracts/integrations" directory. These files are NOT included in the scope of this audit.

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 v0.8.2

Steps taken to run the tools:

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

Findings

QSP-1 Privileged Roles and Ownership

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `contracts/LoanPriceOracle.sol`, `contracts/Vault.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. The account annotated as the `owner` has ultimate access to key variables of the protocol. Similarly, note adapters have extreme privileges related to deciding about the loans' liquidation and repayment status.

Recommendation: (I) Clearly documenting the responsibilities and capabilities of the privileged roles. (II) Initializing the owner variables to trusted entities with options to recover access whenever it is lost. (III) Considering introducing timelocks for key operations.

Update: Acknowledged. Message from MetaStreet team:

> LoanPriceOracle was migrated from Ownable to AccessControl in commit `33f8cd1 contracts: use AccessControl instead of Ownable in LoanPriceOracle`, with the following roles:
>
> - `PARAMETER_ADMIN_ROLE`: can only call `setMinimumDiscountRate()`, `setMinimumLoanDuration()`, `setCollateralParameters()` and will be assigned to governance
> - `DEFAULT_ADMIN_ROLE`: can grant and revoke roles, will be assigned to governance
>
> Vault uses AccessControl with the following roles:
>
> - `COLLATERAL_LIQUIDATOR_ROLE`: can only call `withdrawCollateral()` and `onCollateralLiquidated()`
> - `EMERGENCY_ADMIN_ROLE`: can only call `pause()` and `unpause()`
> - `DEFAULT_ADMIN_ROLE`: can call `setSeniorTrancheRate()`, `setAdminFeeRate()`, `setLoanPriceOracle()`, `setNoteAdapter()`, `withdrawAdminFees()`, can grant and revoke roles, will be assigned to governance
>
> Once the governance system is in place, we will deploy these roles behind a TimeLockController that is controlled by the DAO.
>
> LPToken is Ownable, but is transferred immediately to the associated Vault after creation.
>
> Note Adapters do need to be implemented correctly, but they are essentially library contracts with little to no state. If a bug is discovered in a note adapter, the Vault can be paused by the emergency admin, the note adapter can be upgraded by the default admin, and then Vault operations can be unpaused.

QSP-2 Lack Of Validation On The Loans Sold To The Vault

Severity: *Medium Risk*

Status: Fixed

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

Description: Looking at the implementation of `_sellNote(...)`, there is a lack of explicit checks on the underlying loan of the promissory note that is being sold to the `Vault`.

Recommendation: Either provide an explanation in the white paper for why existing or expired loans can't be resold to the `Vault` or introduce some sort of sanity checks on the underlying loan of the promissory note being sold.

Update: Fixed. Message from MetaStreet team:

> The Note Adapter is responsible for validating the status of a loan in `isSupported()`. This check currently exists in all integration note adapters. A check has been added to TestNoteAdapter in commit `c8c2b96 contracts/test: add loan status check to isSupported() in TestNoteAdapter` (PR #17) to clarify this responsibility of the note adapter.
>
> An expired loan would have reverted from underflow in the `loanTimeRemaining` calculation of `priceLoan()` in the LoanPriceOracle. This check has been refactored to revert more gracefully on an expired loan in `f2be1b1 contracts: refactor minimum loan duration check in LoanPriceOracle`.

QSP-3 Oracle Centralization

Severity: *Medium Risk*

Status: Mitigated

Description: Due to its centralized nature, a vulnerability can arise from depending on a single centralized oracle to deliver pricing data for the loans. A centralized oracle carries the inherent risk of having its private keys compromised, as well as forcing the community to trust that the centralized authority that owns the oracle will not abuse its position of power and submit malicious data.

Recommendation: Consider implementing a decentralized oracle such as Chainlink.

Update: Mitigated. Message from MetaStreet team:

> Loan price oracle parameters will be controlled by a decentralized governance process, in which stakeholders can vote to steer the interest rate components for different collateral

tokens.

> Collateral value is currently updated manually, but would be done so subject to a veto vote by governance. At the moment, we are avoiding the use of an on-chain decentralized oracle, as they do not yet implement our valuation methodology, but this will likely change in the near future.

QSP-4 Concerns On Admin Fee System

Severity: Medium Risk

Status: Fixed

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

Description: Admin fees are only applicable to successful loan repayments. However, going through the logic implemented in `Vault.sol`, when a promissory note is first sold in `_sellNote(...)`, the `adminFeeRate` is applied to the `seniorTrancheReturn` on the following line:

```
L746 seniorTrancheReturn -= PRBMathUD60x18.mul(_adminFeeRate, seniorTrancheReturn);
```

and then saved in the `loan`'s state on the following line:

```
L769 loan.seniorTrancheReturn = seniorTrancheReturn;
```

This `seniorTrancheReturn` is then updated in `onLoadExpired(...)` on the following line:

```
L1005 /* Update senior tranche return for collateral liquidation */
L1006 loan.seniorTrancheReturn += seniorTrancheLoss;
```

Additionally, this updated `seniorTrancheReturn` is then being used in the `onCollateralLiquidated(...)` callback on the following line:

```
L1036 uint256 seniorTrancheRepayment = Math.min(proceeds, loan.seniorTrancheReturn);
```

However, it seems that the `adminFeeRate` has not been factored out of the `seniorTrancheReturn`, even though the loan was unsuccessfully repaid.

Recommendation: Factor out the admin fee from the `seniorTrancheReturn` on unsuccessful loan repayments.

Update: Fixed. Message from MetaStreet team:

> This has been addressed in commit `d5d9832 contracts: move admin fee processing to onLoanRepaid() in Vault` (PR #18), in which the admin fee is only applied in the `onLoanRepaid()` path. The admin fee is no longer implicitly removed from the senior tranche repayment in the loan expired and collateral liquidated paths.

QSP-5 Serviced Loans Are Not Removed From Pending Loans

Severity: Low Risk

Status: Acknowledged

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

Description: After a loan has been serviced by `performUpkeep(...)`, `onLoanRepaid(...)`, `onLoanExpired(...)` or `onCollateralLiquidated(...)` the loan is no longer pending and should be removed from `_pendingLoans` to avoid confusion.

Recommendation: Consider removing the serviced loan from `_pendingLoans`.

Update: Message from MetaStreet team:

> This will not pose an issue in practice, since the loan status will change from Active after processing and will then be filtered out in subsequent calls to `checkUpkeep()`. Removing a loan ID from the `uint256[]` array inside of `_pendingLoans` would require a linear search, which may be large if many loans are maturing in that time bucket. Similarly, changing `_pendingLoans` to use an `EnumerableSet.UintSet` for constant time insertion and removal adds substantial gas costs to the `sellNote()` path, which we would like to keep as inexpensive as possible. Since there isn't a real benefit to removing the loan, and changing the data structure adds substantial gas cost to a critical path, we are opting to leave the loans in the pending loans state.
>
> In addition, the pending loans getter provides an API to enumerate all of a Vault's past and current loans, without relying on historical events.

QSP-6 Missing Input Validation

Severity: Low Risk

Status: Fixed

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

Related Issue(s): SWC-123

Description: `setCollateralParameters(...)` does not check the input parameters of `packedCollateralParameters`. This can lead to wrong loan price computation (e.g. the rate component weights do not sum 1).

Recommendation: We recommend adding the relevant checks.

Update: Fixed. Message from MetaStreet team:

> This has been addressed in commit `d2d8c5c contracts: validate weights in setCollateralParameters() in LoanPriceOracle` (PR #19), which checks that weightings sum to 100. We have refrained from adding any validation to the linear models that might have hardcoded assumptions about their values.

QSP-7 Interaction with External Contracts

Severity: Informational

Status: Acknowledged

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

Description: The protocol relies on functionalities of external contracts. Therefore, the security of the project depends on these contracts. While we are unaware of any immediate issues, it is important to note that external contracts may be vulnerable.
In this audit, the implementation of note adapters (defined by `INoteAdapters.sol`interface) was not covered.

Recommendation: We recommend reviewing external contracts to make sure they work as expected.

Update: MetaStreet team acknowledged this issue.

QSP-8 Race Conditions / Front-Running

Severity: Informational

Status: Acknowledged

File(s) affected: `./contracts/LPToken.sol`, `./contracts/Vault.sol`

Related Issue(s): SWC-114

Description: A block is an ordered collection of transactions from all around the network. It's possible for the ordering of these transactions to manipulate the end result of a block.
It is currently possible for liquidity providers to monitor for upcoming `redeem(...)` transactions and front-run them. This is particularly important for the case when a contract is about to lose value, for instance, due to an expiring loan. Certain actors may "rush" to take up a place in the redeem queue to be able to qualify for a withdrawal sooner.

Recommendation: There is currently no known prevention from it. One likely mitigation is for the liquidity provider (LP) to submit a transaction bypassing the mempool using a third-party service.

Update: MetaStreet team acknowledged this issue. Message from MetaStreet team:

> This situation can occur if a loan is expiring and its collateral value is less than the principal.

QSP-9 Deprecated Function

Severity: Informational

Status: Fixed

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

Description: `Vault` contract inherits from OpenZeppelin's `AccessControlUpgradeable` contract to control access and permissions. `Vault.initialize(...)` calls `_setupRole(...)`, a function being deprecated in favor of `_grantRole(...)`. For further information, visit OpenZeppelin's documentation.

Recommendation: Replace `_setupRole(...)` with `_grantRole(...)`.

Update: Fixed. Message from MetaStreet team:

> This has been addressed in commit `cbda741 contracts: change deprecated _setupRole() to _grantRole() in Vault` (PR #20).

QSP-10 [Not An Issue] Concerns On Upgradability

Severity: Informational

Status: Acknowledged

File(s) affected: `./contracts/LoanPriceOracle.sol`, `./contracts/LPToken.sol`, `./contracts/Vault.sol`

Description: Section `7.1 Proxied Components` in the specification states that the Vault, LP Tokens, and Loan Price Oracle will be deployed as proxied smart contracts. However, the contracts' storage is not optimized for upgradability: it is easy to damage the storage after an upgrade. Additionally, it is recommended to put the upgrade logic behind a timelock for better user trust.

Update: After discussion with MetaStreet team, we decided to label this issue as a false positive.

QSP-11 Presence Of Unused Variables

Severity: Informational

Status: Acknowledged

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

Related Issue(s): SWC-131

Description: Unused variables are allowed in Solidity and they do not pose a direct security issue. It is best practice though to avoid them as they can: (I) cause an increase in computations (and unnecessary gas consumption), (II) indicate bugs or malformed data structures and they are generally a sign of poor code quality, (III) cause code noise and decrease the readability of the code.

This issue was found in `LoanPriceOracle.priceLoan(...)`. It does not use `collateralTokenId` and `duration` variables.

Recommendation: Remove all unused variables from the codebase.

Update: Message from MetaStreet team:

> These are intentionally provided in the generic `priceLoan()` API to support future changes in the loan price oracle that may use the loan duration or specific collateral token IDs. By keeping these arguments in the API, we can upgrade the loan price oracles or deploy custom loan price oracles that support these parameters, without needing to upgrade the Vaults.

QSP-12 Missing Address Validation

Severity: Undetermined

Status: Fixed

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

Description: In the `onLoanExpired(...)` callback, there is no check for zero address on the target address being called in the following lines:

```
(address target, bytes memory data) = noteAdapter.getLiquidateCalldata(loanId);

/* Call liquidate on lending platform */
(bool success, ) = target.call(data);
if (!success) revert CallFailed();
```

Recommendation: Introduce a check for `0x0` address on the `target` address being returned from `noteAdapter.getLiquidateCalldata(loanId)` on `L1012`, as done in `withdrawCollateral(...)`.

Update: Fixed. Message from MetaStreet team:

> This has been addressed in commit `2365243 contracts: add liquidate target validation to onLoanExpired() in Vault` (PR #21).

QSP-13 Zero-Value Transfers

Severity: *Undetermined*

Status: Fixed

File(s) affected: `./contracts/LPToken.sol`, `./contracts/Vault.sol`

Description: It is currently possible to withdraw or request the redemption of the zero amount.

Recommendation: While the implications of it are unclear, it is recommended to disallow this scenario unless it contradicts the intended business logic.

Update: Fixed. Message from MetaStreet team:

> This has been addressed in commits `441d302 contracts: revert on zero amount in deposit() and redeem() in Vault` and `89f7148 contracts: skip processing on zero amount in withdraw() in Vault` (PR #22).

Automated Analyses

Slither

Slither reported 388 results, many of which have been identified as false positives. Relevant results have been incorporated into the findings of this report.

Adherence to Specification

Overall, the white paper was well written and provided thorough insight into the mechanisms implemented in the code. The code, for the most part, matches the provided specification. MetaStreet team also provided a code walkthrough video to explain the architecture and system modules in detail. However, there were several issues that were still present:

- Lack of documentation on the Admin Fee implementation in the white paper.
- Lack of documentation on the note adapter implementation in the white paper.
- The system is only compatible with ERC20 tokens with 18 decimals. Restricted in the code, but not mentioned in the specification.
- The scope provided to us excluded the implementation of `VaultRegistry.sol`, `NFTfiV2NoteAdapter.sol`, and `ArcadeV1NoteAdapter.sol`, covered in the code-walkthrough video.
- `3.1.6 Loan Default`: Could not find the exact corresponding code.
- `3.4.1 Collateral Value`: There is no corresponding code. However, does not look to be in the scope of the audit.
- `4.4 Sell Note` shows some discrepancies:
  - `Validate loan purchase price is greater than minimum purchase price`: in the code, it is "greater than or equal".
  - `Validate Vault has sufficient cash for purchase`: it is unclear whether it is expected to be `purchasePrice > _totalCashBalance` or `purchasePrice >= _totalCashBalance`.
  - `Validate Senior Tranche Return is less than loan return` was not found in the code.
- `4.7 Liquidate Loan`: the code is not present in the scope.
- `4.9 Loan Repaid Callback`: `Total Loan Balance` is not present in the code.
- `4.10 Loan Liquidated Callback`: unclear whether it corresponds to `onLoanExpired(...)`.
- `4.13 Redemption Processing (Internal)`: `updateReservesBalance()` does not exist.
- `4.14 Cash Reserves Processing (Internal)` is not present. Likely, the documentation is out-of-date when matched with the latest commit.
- `5.3 Price Loan`:
  - `Validate loan duration is greater than Minimum Loan Duration`: In the code, it is "greater than or equal".
  - `Compute utilization rate component` and `Compute loan duration rate component`: strictly speaking, these are not being computed but simply fetched.
- Redemption-related logic (such as the logic of `redemptionAvailable(...)`) is not fully documented in the specification.
- `sellNoteAndDeposit(...)` method does not have a corresponding entry in the documentation.
- The unwrapping logic in `withdrawCollateral(...)` is not documented.
- The `view` modifier is sometimes omitted in the method specification, e.g.: `5.1 API`, `priceLoan(...)`.

Code Documentation

The codebase includes inline comments describing the operation of all functions. Also, all functions and structures are commented using the NatSpec format. The project includes a `README` file with instructions on how to run tests, test coverage, and deployment, as well as the tools needed and their compatible versions.

- `README.md` shows `TBD` in `License` section. Consider completing this section.

Adherence to Best Practices

1. Uninitialized variables are assigned with the type's default value. Explicitly initializing a variable with its default value costs unnecessary gas. This can be seen in the following lines of code of `Vault.sol`(**Update:** Optimization done at compilation time. Not modified in order to keep readability.):

```
L559:  for (uint64 i = 0; i < SHARE_PRICE_PRORATION_BUCKETS; i++){
```

```
L1069: for (uint256 i = 0; i < noteTokens.length; i++) {
```

```
L1083: for (uint256 j = 0; j < _pendingLoans[timeBucket][noteToken].length; j++) {
```

2. Caching the array length outside a loop saves reading it on each iteration, as long as the array's length is not changed during the loop. This can be seen in the following lines of code of `Vault.sol` (**Update:** Addressed in `132cdd0 contracts: cache array length outside loops in checkUpkeep() in Vault` (PR #23)):

```
L1069: for (uint256 i = 0; i < noteTokens.length; i++) {
```

```
L1083: for (uint256 j = 0; j < _pendingLoans[timeBucket][noteToken].length; j++) {
```

3. When dealing with unsigned integer types, comparisons with != 0 are cheaper with > 0. This can be seen in the following lines of code of `Vault.sol` (**Update:** Addressed in`99b0af2 contracts: improve zero comparison in sellNoteAndDeposit() in Vault` (PR #24)):

```
L832: if (seniorTrancheAmount > 0) _deposit(TrancheId.Senior, seniorTrancheAmount);

L833: if (juniorTrancheAmount > 0) _deposit(TrancheId.Junior, juniorTrancheAmount);
```

4. Use of `uint64` rather than `uint256` in `_timestampToTimeBucket(uint64 timestamp)` and the related structs seems unnecessary as gas is not being saved. (**Update:** Addressed in `29cfed7 contracts: use uint256 for time bucket computations in Vault` (PR #25). MetaStreet decided to keep casting to `uint64` in some cases to ease interoperation with other external systems.)

Test Results

Test Suite Results

The project includes an adequate test suite. All tests passed.

```
> test
> hardhat test

No need to generate any newer typings.


  Integration
Warning: Potentially unsafe deployment of Vault

    You are using the `unsafeAllow.delegatecall` flag.

    single loan
       ✓ tests loan repayment (319ms)
       ✓ tests loan default with higher liquidation (308ms)
       ✓ tests loan default with lower liquidation (301ms)
    many loans
       ✓ successfully processes random loans (12416ms)
    proxy deployment
       ✓ multicall succeeds (45ms)
       ✓ tests LPToken upgrade (129ms)
Warning: Potentially unsafe deployment of TestVaultUpgrade

    You are using the `unsafeAllow.delegatecall` flag.

       ✓ tests Vault upgrade (188ms)

  LoanPriceOracle
    constants
       ✓ matches implementation version
    constructor
       ✓ fails on unsupported currency token decimals (41ms)
    #priceLoan
       ✓ price loan on utilization component
       ✓ price loan on loan-to-value component
       ✓ price loan on duration component
       ✓ price loan on all components
       ✓ price loan for zero repayment
       ✓ fails on insufficient time remaining
       ✓ fails on unsupported token contract
       ✓ fails on parameter out of bounds (utilization)
       ✓ fails on parameters out of bounds (loan to value)
       ✓ fails on parameter out of bounds (duration)
    #setCollateralParameters
       ✓ sets collateral parameters successfully (43ms)
       ✓ replaces collateral parameters successfully (47ms)
       ✓ fails on invalid address
       ✓ fails on invalid caller
    #setMinimumDiscountRate
       ✓ sets minimum discount rate successfully
       ✓ fails on invalid caller
    #setMinimumLoanDuration
       ✓ sets minimum loan duration successfully
       ✓ fails on invalid caller

  LPToken
    constants
       ✓ matches expected implementation
    #mint
       ✓ succeeds in minting tokens
       ✓ fails on invalid caller
    #redeem
       ✓ succeeds on valid tokens
       ✓ fails on insufficient tokens
       ✓ fails on outstanding redemption
       ✓ fails on invalid caller
    #withdraw
       ✓ succeeds on full withdrawal
       ✓ succeeds on partial withdrawals
       ✓ fails on excessive amount
       ✓ fails on invalid caller
    #redemptionAvailable
       ✓ returns zero on no redemption pending
       ✓ returns full amount on full redemption available
       ✓ returns partial amount on partial redemption available
       ✓ returns zero on no redemption available

  TestLendingPlatform
     ✓ lend and repay (111ms)
     ✓ lend and liquidate (92ms)

  Vault Accounting
    tranche returns and realized value
       ✓ only senior tranche (158ms)
       ✓ only junior tranche (155ms)
       ✓ increase from repayment (248ms)
       ✓ decrease from default, only junior tranche (182ms)
       ✓ decrease from default, both junior tranche and senior tranche (187ms)
       ✓ collateral liquidation, junior tranche recovery (204ms)
       ✓ collateral liquidation, senior tranche recovery (202ms)
       ✓ increase from liquidation, appreciation (181ms)
    share price appreciation
       ✓ share price appreciation from one loan (360ms)
```

```
      ✓ share price appreciation from overlapping loan maturities (455ms)
    redemption
      ✓ order is senior tranche, junior tranche, undeployed cash (325ms)
    admin fees
      ✓ loan repaid with admin fee (120ms)
      ✓ loan liquidated with admin fee (120ms)
      ✓ collateral liquidated with admin fee, lower liquidation (143ms)
      ✓ collateral liquidated with admin fee, break even (142ms)
      ✓ collateral liquidated with admin fee, higher liquidation (142ms)

  Vault Keeper Integration
    #checkUpkeep
      ✓ detects repaid loan (71ms)
      ✓ detects expired loan (62ms)
      ✓ detects repaid loan in previous time bucket (81ms)
    #performUpkeep
      ✓ services repaid loan (80ms)
      ✓ services expired loan (80ms)
      ✓ fails on invalid code
    consecutive upkeeps
      ✓ handles multiple loans (337ms)

  Vault
    constants
      ✓ matches implementation version
    #initialize
      ✓ fails on invalid addresses
      ✓ fails on unsupported currency token decimals (48ms)
    initial state
      ✓ getters are correct
      ✓ roles are correct
      ✓ tranche states are initialized (38ms)
    #deposit
      ✓ deposits into senior tranche (44ms)
      ✓ deposits into junior tranche (41ms)
      ✓ fails on junior tranche insolvency (107ms)
      ✓ fails on senior tranche insolvency (102ms)
      ✓ fails on insufficient funds
    #deposit (multicall)
      ✓ deposits into both tranches (63ms)
      ✓ fails on reverted call
      ✓ fails on invalid call
    #sellNote
      ✓ sells note (111ms)
      ✓ fails on unsupported note token
      ✓ fails on unsupported note parameters
      ✓ fails on low purchase price
      ✓ fails on high purchase price
      ✓ fails on insufficient cash
      ✓ fails on low senior tranche return (60ms)
    #sellNoteAndDeposit
      ✓ sells note and deposits (115ms)
      ✓ sells note and deposits to only senior (79ms)
      ✓ sells note and deposits to only junior (79ms)
      ✓ fails on low purchase price (60ms)
      ✓ fails on invalid allocation
    #redeem
      ✓ redeems (86ms)
      ✓ redemption scheduled after cash drained (118ms)
      ✓ redemption on insolvent tranche is delayed (148ms)
      ✓ immediate redemption causing insolvent tranche succeeds (138ms)
      ✓ immediate redemption alongside insolvent tranche succeeds (138ms)
      ✓ redemptions processed from new deposit (140ms)
      ✓ fails on invalid amount
      ✓ fails on outstanding redemption
      ✓ fails on junior tranche insolvency (104ms)
      ✓ fails on senior tranche insolvency (106ms)
    #withdraw
      ✓ withdraws successfully (163ms)
      ✓ immediate withdraws successfully (79ms)
      ✓ partial withdraws successfully (119ms)
      ✓ withdraws maximum available (121ms)
      ✓ withdraws maximum available after several withdraws (143ms)
    #withdrawCollateral
      ✓ withdraws collateral after liquidation (121ms)
      ✓ fails on unliquidated loan (72ms)
      ✓ fails on already withdrawn collateral (126ms)
      ✓ fails on unknown loan
      ✓ fails on invalid caller
    #onLoanRepaid
      ✓ succeeds on repaid loan (130ms)
      ✓ fails on unrepaid loan (87ms)
      ✓ fails on repaid loan with callback processed (105ms)
      ✓ fails on liquidated loan (78ms)
      ✓ fails on liquidated loan with callback processed (99ms)
      ✓ fails on liquidated loan with callback processed and collateral withdrawn (227ms)
      ✓ fails on liquidated collateral with callback processed (135ms)
      ✓ fails on unknown loan
      ✓ fails on unsupported note
    #onLoanExpired
      ✓ succeeds on expired loan (138ms)
      ✓ fails on repaid loan (178ms)
      ✓ fails on repaid loan with callback processed (103ms)
      ✓ fails on liquidated loan with callback processed (100ms)
      ✓ fails on liquidated loan with callback processed and collateral withdrawn (115ms)
      ✓ fails on liquidated collateral with callback processed (127ms)
      ✓ fails on unknown loan
      ✓ fails on unsupported note
    #onCollateralLiquidated
      ✓ succeeds on liquidated loan (150ms)
      ✓ fails on unliquidated loan (71ms)
      ✓ fails on repaid loan (87ms)
      ✓ fails on repaid loan with callback processed (104ms)
      ✓ fails on liquidated collateral with callback processed (125ms)
      ✓ fails on unknown loan
      ✓ fails on invalid caller
    #utilization
      ✓ achieves utilization of 25% (59ms)
      ✓ achieves utilization of 50% (59ms)
      ✓ achieves utilization of 100% (58ms)
    #pendingLoans
      ✓ returns loans pending across two time buckets (193ms)
    #withdrawAdminFees
      ✓ withdraws admin fees successfully (119ms)
      ✓ fails on invalid address
      ✓ fails on invalid amount
      ✓ fails on invalid caller
    #setSeniorTrancheRate
      ✓ sets senior tranche rate successfully
      ✓ fails on invalid value
      ✓ fails on invalid caller
    #setAdminFeeRate
      ✓ sets admin fee rate successfully
      ✓ fails on invalid value
      ✓ fails on invalid caller
    #setLoanPriceOracle
      ✓ sets loan price oracle successfully
      ✓ fails on invalid address
      ✓ fails on invalid caller
    #setNoteAdapter
      ✓ sets note adapter successfully
      ✓ fails on invalid address
      ✓ fails on invalid caller
    #pause/unpause
      ✓ pauses and unpauses
      ✓ deposit fails when paused
      ✓ sell note fails when paused
      ✓ sell note and deposit fails when paused
      ✓ redeem fails when paused
      ✓ withdraw fails when paused
      ✓ fails on invalid caller (46ms)
    #supportsInterface
      ✓ returns true on supported interfaces
      ✓ returns false on unsupported interfaces


  166 passing (27s)
```

Code Coverage

Code coverage shows good metrics with a high percentage of coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	97	100	100	
LPToken.sol	100	100	100	100	
LoanPriceOracle.sol	100	100	100	100	
Vault.sol	100	96.05	100	100	
contracts/interfaces/	100	100	100	100	
ILoanPriceOracle.sol	100	100	100	100	
ILoanReceiver.sol	100	100	100	100	
INoteAdapter.sol	100	100	100	100	
IVault.sol	100	100	100	100	
All files	**100**	**97**	**100**	**100**	

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

```
aa4c5460ad4d8c4b0d95ff6a0cd6637a70375ec83e3f6ebe5f2e91861478e871 ./contracts/Vault.sol
f7d3225665a2c1a03b8edb389e8bc1a193a3bd39f9772b2fc61588176830292b ./contracts/LPToken.sol
7670af7004b247c4fdad716ec278a0c8ee9985ce37f53acf061927acafeb45c2 ./contracts/LoanPriceOracle.sol
f49c7fb192f6a9fa3c14a809a793a9b9b8dff9b9e17b82876c634152d4ca94a7 ./contracts/interfaces/IVault.sol
a65eca0279b0bc72a8d63d885ced649fd5792c705136426f731a1a9894bc30e5 ./contracts/interfaces/INoteAdapter.sol
d3c55e587f3fcbee6cbf1dede558130143191bf92c3c7aa3b3331737fe092055 ./contracts/interfaces/ILoanPriceOracle.sol
309290098fa69908d0ef1adb079ea09e1a949ad0fc222659ced5f211917c211d ./contracts/interfaces/ILoanReceiver.sol
```

Tests

```
2e5b11e886aa2cdd220b12a95f069f6d910d511fdc12aa7fc6313d294203e762 ./contracts/test/TestLPTokenUpgrade.sol
9c1e70b59935bba75498803f1c8e8e58e083c14dab40b41c2d0a467b1bbe523d ./contracts/test/TestERC721.sol
c9cf70742740f72526648634eabdb248cab10f69019ae8245dbd9a40e856ded4 ./contracts/test/MockLoanPriceOracle.sol
10ab3efd884a568e4840b3846871675c6ee909fd169a7c68b49f4a5836f919e5 ./contracts/test/TestVaultUpgrade.sol
93dece3b90d621a3270ac90a3628ed645081dc39e06db9cc14667d2bc0cd049b ./contracts/test/TestERC20.sol
213a2eae8599d4b7e5af35a31d2b55076c5ed4faee07f56f8e60191d6ab04a1b ./contracts/test/lending/TestNoteAdapter.sol
59b81bf63a1b7c919273de350d62f5672f443885410bcd481c152559a47358d5 ./contracts/test/lending/TestNoteToken.sol
3e541a1543b91f42fb191d04f519893d789a8b53a271dde21f59e5e194ec50a0 ./contracts/test/lending/TestLendingPlatform.sol
1aad405d7981b6a1f9ee2598af3f0c7d5f14a3cf7e03bb67c4ceeb825bebf93e ./test/LoanPriceOracle.spec.ts
d250570971e27eda8b0001f84406545b173c5e4be25adb49c80ea5c27a0413d9 ./test/TestLendingPlatform.spec.ts
0cb5f8a16fc22b558b8734e4c900ddca29c59b554f1a57cd9d9869a8b37b3d0d ./test/Vault.accounting.spec.ts
6cb8ae5509bfc7f8260e12f99e33f6de6b80ca3edfc1b7d5737b2e7c217944ea ./test/Vault.spec.ts
64d663567c369ce9f75b727243327d275413810989ad50b8d12c589b63e7d14c ./test/Vault.keeper.spec.ts
52bebf535824baf1b6168a64f6ee4df7c0b02498904267ce0bc5f59c200fa8ab ./test/Integration.spec.ts
648a821ff5ae405839fea8048a8d39ce81e480baddde02da07510760c43606ec ./test/LPToken.spec.ts
cddccf800b5046e3bc8d142243b9b528634696b0334d5da49fac9633865459e1 ./test/helpers/FixedPointHelpers.ts
408edd7eecad9e232138f95b9ac7251dddc5102475411e29f7aa66769d42bfec ./test/helpers/EventUtilities.ts
f141b1a2fc03a1c800e4f5b388e56a4b466874b02eb1e4e4612312559e97c5a5 ./test/helpers/VaultHelpers.ts
1fdd9e5b668038b6cfc9d783a9dc73ba9d9cbee729925ed3379bbb5e4e220315 ./test/helpers/LoanPriceOracleHelpers.ts
0cd709bc177f597235a9ae9b2a9e900cdf10b1b483776adb0f9a0ea9bd527d5b ./test/helpers/RandomHelpers.ts
```

Changelog

- 2022-04-22 - Initial report
- 2022-05-03 - 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.