Quantstamp Contract Security Certificate

March 20th 2020 — Quantstamp Verified

Lendroid Protocol version 2.0

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Audit
Auditors	Poming Lee, Research Engineer Martin Derka, Senior Research Engineer Sebastian Banescu, Senior Research Engineer Alex Murashkin, Senior Software Engineer
Timeline	2020-01-13 through 2020-03-20
EVM	Istanbul
Languages	Vyper
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Smart Contract - documentation

Source Code	
Repository	Commit
protocol.2.0	af4be80
protocol.2.0	0b414c4
protocol.2.0	eb2ab78

Changelog	• 2020-01-31 - Initial report • 2020-02-24 - Updated report according to commit hash `0b414c4` • 2020-03-20 - Updated report according to commit hash `eb2ab78`

Overall Assessment

Overall, the contracts exhibit a large amount of centralization of power. Accidentally wrong or malicious updates to system components can result in partial or complete malfunction of the platform. The business logic of the system is very complex. While the lengthy documentation, partially created during the audit, is helpful, the size and complexity of the system makes it difficult to assess the correctness of the implementation. It is imperative that the correctness of the implementation be confirmed by unit and integration tests with overall test coverage exceeding 95%, and nearing 100% as much as possible.

We identified 22 findings, one of them was deemed as high-severity, one - as medium-severity, and three - as low-severity. Six findings were marked as "undetermined" since it is not feasible to assess severity given the available data. The remaining findings were marked as informational. It is noted that there were no known Vyper-related vulnerabilities detected.

2020-02-24 update: the Lendroid team has resolved 14 issues. The team acknowledged and made comments on 6 issues. Currently the code is not production-ready due to low test coverage and two unresolved issues. Quantstamp would advise Lendroid team to increase the branch coverage above 80% before going live.

2020-03-18 update: the Lendroid team has resolved 15 issues. The team acknowledged and made comments on 7 issues. The Lendroid team stated that they will increase the branch coverage to 80% before the project go live (currently the branch coverage is 65.14%). Quantstamp advises against going live with the current project due to the current branch-coverage level of under 80%. The Lendroid team should be aware that functionality bugs that we could not check during the audit due to missing details in the technical specification, may appear due to the missing 20% test coverage.

Total Issues	**22**	(15 Resolved)
High Risk Issues	**1**	(1 Resolved)
Medium Risk Issues	**1**	(1 Resolved)
Low Risk Issues	**3**	(2 Resolved)
Informational Risk Issues	**11**	(8 Resolved)
Undetermined Risk Issues	**6**	(3 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.

Summary of Findings

ID	Description	Severity	Status
QSP-1	Inverted Comparison Sign	High	Resolved
QSP-2	Erroneous Implementation of `if-else` statement	Medium	Resolved
QSP-3	Missing Position Unlock	Low	Resolved
QSP-4	Underlying Value Potentially Transferred Twice	Low	Resolved
QSP-5	Allowance Double-Spend Exploit	Low	Acknowledged
QSP-6	Centralization of Power	Informational	Acknowledged
QSP-7	'Dead' Code	Informational	Resolved
QSP-8	Unused Constant	Informational	Resolved
QSP-9	Constants Need Double-Checking	Informational	Resolved
QSP-10	Missing Token Id Checking	Informational	Resolved
QSP-11	Missing check in `safeTransferFrom` function	Informational	Resolved
QSP-12	Misaligned Code Comments and Implementation	Informational	Resolved
QSP-13	Possible Transfer to 0x0 / Contract Address	Informational	Resolved
QSP-14	Pool Owner Can Borrow Money and Change I-Token Cost of the Pool	Informational	Acknowledged
QSP-15	Repeatedly Settable Values	Informational	Acknowledged
QSP-16	Missing Argument Validation	Informational	Resolved
QSP-17	A Possible Way of Implementing Undesirable Action	Undetermined	Acknowledged
QSP-18	Missing Expiry Check	Undetermined	Resolved
QSP-19	Missing Return Statement	Undetermined	Resolved
QSP-20	Timestamp Dependency	Undetermined	Acknowledged
QSP-21	Race Conditions / Front-Running	Undetermined	Acknowledged
QSP-22	Erroneous Assumption of Token Balance	Undetermined	Resolved

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the Lendroid Protocol version 2.0 repository for security-related issues, code quality, and adherence to specification and best practices. From the blockchain security standpoint, Quantstamp assessed all of the aspects with the following result (additional individual findings are listed in the Assessment section):

1. **Transaction Ordering Dependence**
   Transaction ordering dependence in the system exists, however, it is difficult to assess its impact on the security of the system. The system contains a number of configuration parameters and actions that the DAOs, as well as general users, can perform. Every such action requires submitting a transaction which is visible to the entire network in mempool before it is mined. This can be potentially exploited through racing by users reacting to actions such as `close_position` (`MarketDao#L898`) or `set_price_oracle` (`MarketDao#L515`). The severity of such exploits should be determined by the Lendroid team.
2. **Timestamp Dependence**
   The system depends on timestamps. The `Expiry` struct inside `ProtocolDao` uses a timestamp, and so do some configurations of the other DAOs. The Lendroid team should note that timestamps can be influenced by miners within a range of approximately 30 seconds.
3. **Mishandled Exceptions and Call Stack Limits**
   The system does not exhibit signs of mishandled exceptions as no non-reverting calls to external contracts are present.
4. **Unsafe External Calls**
   The system contains a number of external calls whose return values are ignored. This includes mostly initializations and setters. Quantstamp did not discover any scenarios that could lead to an inconsistent state caused by a failed external call that would not revert the transaction. However, it must be noted that the system relies on an interaction of a number of components making externals calls from one to another, and there is no mechanism that can ensure that a component residing on an external address (1) implements the expected interface and supports the desired function call; and (2) has honest implementation and performs the expected action. While Quantstamp did not discover any unsupported external call, some of the components are repeatedly settable, and there is no way of guaranteeing consistency in the future. Additionally, the functional correctness of the current implementation should be confirmed by tests.
5. **Integer Overflow and Underflow**
   The system is not vulnerable to overflows and underflows as guaranteed by Vyper.
6. **Number Rounding Errors**
   We did not discover number rounding errors, but we recommend that tests confirm that any calculations adhere to specification.
7. **Reentrancy and cross-function vulnerabilities**
   The contracts do not contain reentrancy vulnerabilities as all the calls are made within trusted contracts inside the platform. No calls to untrusted contracts are made. However, the caveat of future honest implementation of repeatedly settable components applies.
8. **Denial of service/logical oversights**
   We did not identify denial of service in the system. The assertions, which could cause such behaviour, do not "over-validate" the state, and are merely restricted to access control and pausing of the system.
9. **Access Control**
   Access control exists and appears to be implemented correctly.
10. **Centralization of Power**

The contracts exhibit a large amount of centralization of power. A number of components are repeatedly settable by certain roles. Accidentally wrong or malicious updates to system components can result in partial or complete malfunction of the platform.

11. **Business logic contradicting the specification**
    We made a significant effort to verify the business logic. The issues identified during the audit are listed in the Assessment section. Due to the size of the platform and the underlying logic, we strongly recommend that the functionality is confirmed by unit tests aiming for exceptional coverage.
12. **Code clones, functionality duplication**
    The code appears well organized. We did not identify clones or duplication of functionality.
13. **Gas usage**
    There are a few for-loops that have 1000 iterations. It is not clear to us if the Lendroid team performed a proper gas analysis to be sure that the protocol will not get a DoS via the "out of gas" error. The loops are the following:

    - `L366` in `contracts/daos/MarketDao.v.py`
    - `L405` in `contracts/templates/InterestPoolTemplate1.v.py`
    - `L492` in `contracts/templates/UnderwriterPoolTemplate1.v.py`
14. **Arbitrary token minting**
    Arbitrary token minting issues are restricted to privileged and trusted roles. Other than arbitrary token minting, the contracts contain draining function (so-called "escape hatches") for emergency interventions.
15. **Parameter validation**
    Many of the functions lack parameter validation. This includes initializations, setters, as well as other user actions. Validation of parameters should be added.

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.

Assessment

Findings

QSP-1 Inverted Comparison Sign

Severity: *High Risk*

Status: Resolved

File(s) affected: `InterestPoolTemplate1.v.py`, `UnderwriterPoolTemplate1.v.py`

Description: `if _expiry >= block.timestamp` compares the timestamp in the wrong direction. This issue occurs:

- on `L118`, InterestPoolTemplate1.v.py
- on `L132`, UnderwriterPoolTemplate1.v.py

More details: The comparison on `L118` is checking if the current timestamp is before the expiry timestamp and if so it is returning zero (`L119`) for the value of the i token. This is incorrect as the value of the i token is zero after the loan market expires, not before.

Recommendation: The sign needs to be flipped.

QSP-2 Erroneous Implementation of `if-else` statement

Severity: *Medium Risk*

Status: Resolved

File(s) affected: `InterestPoolDao.v.py`, `ShieldPayoutDao.v.py`

Description: On `L404-L413` of `InterestPoolDao.v.py`, `_token` will always be `ZERO_ADDRESS`. On `L326-333` of `ShieldPayoutDao.v.py`, `_token` will always be `ZERO_ADDRESS`.

Recommendation: Assign correct values to `_token`.

QSP-3 Missing Position Unlock

Severity: *Low Risk*

Status: Resolved

File(s) affected: `PositionRegistryTemplate1.v.py`

Description: `L262` locks the `_position_id`, however, this is never unlocked in the context of the same function `close_liquidated_loan`, which is probably unintended.

Recommendation: Change the `self._lock_position(_position_id)` on `L275` to `self._unlock_position(_position_id)`.

QSP-4 Underlying Value Potentially Transferred Twice

Severity: *Low Risk*

Status: Resolved

File(s) affected: `SimpleCollateralAuctionCurveTemplate1.v.py`

Description: Both `L165` and `L167` transfer the `_underlying_value` by calling `_transfer_f_underlying` with different destination addresses.

Recommendation: Check if it is meant to transfer the `_underlying_remaining` value on `L167` instead of the `_underlying_value`.

QSP-5 Allowance Double-Spend Exploit

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ERC20Template1.v.py`, `ERC20PoolTokenTemplate1.v.py`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit
See the `approve` (`ERC20Template1.v.py`, `L111`) and `_approve` (`ERC20Template1.v.py`, `L96`) functions.
Same holds for the `ERC20PoolTokenTemplate1.v.py` file with the `approve` (`L119`) and `_approve` (`L104`) functions.

Exploit Scenario: Allowance double-spend exploit example:
(https://github.com/OpenZeppelin/openzeppelin-solidity/blob/b4f87bb8fc25fb07f73099701e39e167a3d36465/contracts/token/ERC20/ERC20.sol#L71-L78), as with other ERC20 tokens. An example of an exploit goes as follows:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and `decreaseAllowance`.
Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.
2020-02-15 update: The Lendroid team stated that they "will await community agreement for ERC Standard."

QSP-6 Centralization of Power

Severity: *Informational*

Status: Acknowledged

File(s) affected: `MultiFungibleTokenTemplate1.v.py`, `PoolNameRegistryTemplate1.v.py`, `ERC20PoolTokenTemplate1.v.py`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.
In the case of the Lendroid protocol, the governor role holds a certain level of centralized power over the system. It can change the role of the other players, the system parameters, and the template contracts the system is using, after the contracts are deployed and go live.
In `MultiFungibleTokenTemplate1.v.py`, there are several functions that enable authorized DAOs to perform actions like: `safeTransferFrom`, `burn`, `mint`, on behalf of another account.
In `PoolNameRegistryTemplate1.v.py`, the `escape_hatch_erc20` function defined on `L270` can transfer all ERC20 tokens of a given `_currency` address to an escape hatch token holder and is callable only by the Protocol DAO.
In `ERC20PoolTokenTemplate1.v.py`, `L159`: `mintAndAuthorizeMinter` and `L209`: `burnAsAuthorizedMinter`: contract owner can mint and burn whatever amount of token they want.

Recommendation: We suggest informing the community of the privileged system actor roles.
2020-02-15 update: The Lendroid team stated that "Documentation to inform the community is underway."

QSP-7 'Dead' Code

Severity: *Informational*

Status: Resolved

File(s) affected: `CurrencyDao.v.py`

Description: "Dead" code refers to code whose execution makes no impact on the final result. Dead code raises a concern, since either the code is unnecessary or the necessary code's results were ignored. Regardless, further investigation is required.
`L613` defined function `authorized_unwrap()` and can only be called by `MarketDAO`. However, we cannot find any calling of this function in the `MarketDAO` contract.

Recommendation: Remove the unnecessary code.

QSP-8 Unused Constant

Severity: *Informational*

Status: Resolved

File(s) affected: `MultiFungibleTokenTemplate1.v.py`

Description: `MFT_ACCEPTED` is an unused private constant.

Recommendation: Quantstamp recommends clarifying the constant's use, or removing the code.

QSP-9 Constants Need Double-Checking

Severity: *Informational*

Status: Resolved

File(s) affected: `MultiFungibleTokenTemplate1.v.py`

Description: The hash on `L73` may be incorrect. In the comments, there are discrepancies with the actual method names: "hash" in the comments is really `_hash` in the actual contract.

Recommendation: Quantstamp recommends veryfing the listed, as well as all related, values.

QSP-10 Missing Token Id Checking

Severity: *Informational*

Status: Resolved

File(s) affected: `MultiFungibleTokenTemplate1.v.py`

Description: Function `mint()` on `L215` does not check if the token with the given `id` is present.

Recommendation: Quantstamp recommends validating the existence of the token id.

QSP-11 Missing check in `safeTransferFrom` function

Severity: *Informational*

Status: Resolved

File(s) affected: `MultiFungibleTokenTemplate1.v.py`

Description: Function `safeTransferFrom()` is not implemented according to the in-code documentation: the standard requires checking if `_to` is a contract.

QSP-12 Misaligned Code Comments and Implementation

Severity: *Informational*

Status: Resolved

File(s) affected: `contracts/daos/ProtocolDao.vy`, `contracts/templates/InterestPoolTemplate1.v.py`, `contracts/templates/PositionRegistryTemplate1.v.py`

Description: The code contains the following discrepancies:

1. `contracts/daos/CurrencyDao.v.py`: The comment on `L44` specifies `# template_name => template_contract_address`. However, on `L45`, we see a mapping from signed integer (not a string) to an address: `templates: public(map(int128, address))`. The comment or the implementation be updated.
2. `contracts/templates/InterestPoolTemplate1.v.py`: `L249` contains a code comment that says: `# verify mft_expiry_limit_days has been set`. However, this aspect is not verified on the subsequent lines of code. It should be checked however via an assert statement such as `assert self.mft_expiry_limit_days > 0`.
3. `contracts/templates/PositionRegistryTemplate1.v.py`: `L32` contains a code comment that says: `# expiry => (loan_id => Loan)`. However, the following line contains a 1-dimensional mapping `positions: public(map(uint256, Position))`, which only seems to hold `(loan_id => Loan)`. The comment should be updated.
4. `contracts/daos/ProtocolDao.vy`: `L288, L305` (of commit `0b414c4`) the comment is incorrect "to change the Governor".

Recommendation: Fixing these comments.
2020-02-15 update: The Lendroid team have fixed the items 1-3. However, item 4 `L288` and `L305` in `ProtocolDao.vy` (of commit `0b414c4`) still requires a fix.

QSP-13 Possible Transfer to 0x0 / Contract Address

Severity: *Informational*

Status: Resolved

File(s) affected: `ERC20Template1.v.py`, `ERC20PoolTokenTemplate1.v.py`

Description: It is rarely desirable for tokens to be sent to the `0x0` address (intentional token burning is a notable exception) nor to the contract itself. However, these mistakes are often made due to human errors. Hence, it's often a good idea to prevent these mistakes from happening within the smart contract itself.

Recommendation: Require that `_to` is different from `0x0`.

QSP-14 Pool Owner Can Borrow Money and Change I-Token Cost of the Pool

Severity: *Informational*

Status: Acknowledged

File(s) affected: `InterestPoolTemplate1.v.py`

Description: There is nothing stopping the pool owner from borrowing from a pool if the pool owner accepts public contributions and there are many liquidity providers. The pool owner can borrow all the funds and then change the cost of i-tokens to 0 such that they do not pay any interest for all the borrowed funds.

Recommendation: Prevent pool owners from borrowing funds. However, the pool owners can just use another account and still do this.
2020-02-15 update: The Lendroid team stated that "This is intentional as Malicious for adversarial behaviors (market will learn not to contribute to their pools)."

QSP-15 Repeatedly Settable Values

Severity: *Informational*

Status: Acknowledged

Description: Some components of the system are repeatedly settable. Changing configuration parameters can affect user funds. Changing system components can affect system behavior, especially since it is not possible to ensure that the newly set components support the required interface and have honest implementation.
The setters can be listed by "`egrep -n 'def set_' -R .`" so we are not including the complete list here, just the summary number of occurrences:

```
CurrencyDao.v.py: 2
ProtocolDao.v.py: 15
MarketDao.v.py: 8
UnderwriterPoolDao.v.py: 4
InterestPoolDao.v.py: 4
```

Recommendation: We suggest documenting the possibility of certain values being set repeatedly.
2020-02-15 update: The Lendroid team stated that "Recommended documentation is underway."

QSP-16 Missing Argument Validation

Severity: *Informational*

Status: Resolved

Description: All DAOs, as well as many of the templates use initializers and setters without validating the arguments.

Recommendation: We recommend adding at least basic validation to all the methods.

QSP-17 A Possible Way of Implementing Undesirable Action

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `InterestPoolDao.v.py`, `UnderwriterPoolDao.v.py`

Description: On `L549` of `InterestPoolDao.v.py` in `deregister_mft_support`, the function just releases the staked LFT and never adjusts the system parameters related to MFT support. It is unclear if this is intentional. Anyone who wants to add `mft_support` can just call `register_mft_support()`, stake LFTs required and withdraw these LFTs immediately right after calling `register_mft_support()`.
On `L585` of `UnderwriterPoolDao.v.py` in `deregister_mft_support`, the function just releases the staked LFT and never adjusts the system parameters related to MFT support (and also never closes the opened `shield_market`). It is unclear if this is intentional. Anyone who wants to add `mft_support` can just call `register_mft_support()`, stake LFTs required and withdraw these LFTs it immediately right after calling `register_mft_support()`.

Recommendation: Adding an additional restriction to the current design of the unstaking methods. For instance, add a timelock.
2020-02-15 update: The Lendroid team has stated that the observed behaviour is intentional.

QSP-18 Missing Expiry Check

Severity: *Undetermined*

Status: Resolved

File(s) affected: `UnderwriterPoolDao.v.py`

Description: `L629`, `L667` did not check `_expiry` of the i-token.

Recommendation: Check `_expiry` of the i-token.

QSP-19 Missing Return Statement

Severity: *Undetermined*

Status: Resolved

File(s) affected: `MultiFungibleTokenTemplate1.v.py`

Description: Documentation on `L148` says `@return Bool` indicating if the id is registered to a hash, but it is not the case as the method does not return anything.

Recommendation: Add the desirable return value or update the comment.

QSP-20 Timestamp Dependency

Severity: *Undetermined*

Status: Acknowledged

Description: The entire system depends on timestamps. Timestamps are manipulable by miners.

Recommendation: Documenting the dependency on timestamps. The Lendroid team should note that timestamps can be influenced by miners within a range of approximately 30 seconds. For example, a miner can choose to regard a market as expired 30 seconds before or 30 seconds after the expiry time set by the owner, which may open room for potential manipulation.
2020-02-15 update: The Lendroid team states that up to 30 seconds of market expiration uncertainty is not considered an issue.

QSP-21 Race Conditions / Front-Running

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `MarketDao.v.py`

Description: A block is an ordered collection of transactions from all around the network. It is possible for the ordering of these transactions to manipulate the end result of a block. A miner attacker can take advantage of this by generating and moving transactions in a way that benefits themselves.
The system contains a number of configuration parameters and actions that the DAOs, as well as general users, can perform. Every such action requires submitting a transaction which is visible to the entire network in mempool before it is mined. This can be potentially exploited by racing by users reacting to actions such as `close_position` (`MarketDao#L898`) or `set_price_oracle` (`MarketDao#L515`).

Recommendation: The severity of such exploits needs to be determined by the Lendroid team taking into account the business logic of the system.
2020-02-15 update: The Lendroid team stated that "We will take steps to inform the community."

QSP-22 Erroneous Assumption of Token Balance

Severity: *Undetermined*

Status: Resolved

File(s) affected: `InterestPoolTemplate1.v.py`

Description: There seems to be an assumption that the token balance is only increased through the `contribute()`. In reality, it could be the case that some tokens are transferred to the address directly (through using ERC20 methods), bypassing that. This could impact the calculations on `L112` and `L146`.

Recommendation: It is recommended to introduce another state variable for keeping track of the balance that was actually contributed.

Code Documentation

The business logic of the system is complex. Most functions in the `templates` folder do not have the minimal necessary code comments, namely short description of purpose and param description. The documentation was mostly created during the audit, and it is rather extensive. We recommend continuing the documentation work and compiling a proper technical document. The level of documentation in the code is inconsistent. Some functions are documented very well, but those are mostly initializers and configuration setters. The business logic functions lack in-code documentation. This level should be unified and additional documentation should be added.

Adherence to Best Practices

• Is `_creator` parameter needed for `_mint` function declared on `L203` of `MultiFungibleTokenTemplate1.v.py`? Its value is only used when emitting the event, which could be emitted inside the `mint` function declared on `L215` as well.

• The maximum allowed length of the name and symbol for the different ERC20 token contracts defined in the `templates/` folder are different. It would be helpful to indicate the significance of the length value using code comments on:

- `L14-15` of `contracts/templates/LERC20Template1.v.py`
- `L14-15` of `contracts/templates/ERC20PoolTokenTemplate1.v.py`
- `L12-13`of `contracts/templates/ERC20Template1.v.py`

• Not clear why the `if` statement inside of the `_remove_name` function (`L129` of `PoolNameRegistryTemplate1.v.py`) is needed. Code comments would help clarify.

• Unnamed constant values are being used on `L155` and `L251` in `InterestPoolTemplate1.v.py` to compute the number of seconds in a day. This should be replaced with a named constant. Same applies to `L179` and `L189` in `UnderwriterPoolTemplate1.v.py`.

• Is the `_estimated_pool_share_tokens` function on `L160` of `InterestPoolTemplate1.v.py` referring to the price of 1 pool share token? If so, then the name of the function should be slightly changed to reflect that.

• `L45-47` in `PositionRegistryTemplate1.v.py` contain 3 declarations of upper-case variables which resemble constants. However, they are not declared using the `constant` keyword, which is inconsistent with the constant declared on `L43` in the same file. Same applies to `L51-52` in `UnderwriterPoolTemplate1.v.py` and `L48` in `InterestPoolTemplate1.v.py`.

• The value `10 ** 18` is being used in multiple locations in the code, e.g. `L188, L200, L203, L217, L219` in `contracts/templates/SimpleCollateralAuctionCurveTemplate1.v.py`. This value should be replaced by a named constant whenever it occurs.

• It is unclear if all the templates that are used in Lendroid will be provided by the Lendroid team. If the team plans on letting the users customize their own templates (based on the interfaces but implemented themselves), it may be risky to include such customized code into the system. The decision of which templates will be use in practice is a responsibility of the governor.

• (for commit `0b414c4`)The unnammed constant value "18" is being used multiple times, e.g. `L64` and `L106` in `UnderwriterPoolTemplateV1.vy`. Should be replaced with a named constant (e.g., `DECIMALS`).

Test Results

Test Suite Results

To summarize, 144 tests passed as of commit `eb2ab78`.

```
brownie test --coverage
Brownie v1.5.1 - Python development framework for Ethereum

Compiling contracts...
  Vyper version: 0.1.0-b16
Generating build data...
 - ERC20Template1...
 - MarketDao...
 - UnderwriterPoolTemplate1...
 - CurrencyDao...
 - InterestPoolDao...
```

```
 - ERC20TokenPoolTemplate1...
 - PoolNameRegistryTemplate1...
 - SimpleCollateralAuctionCurveTemplate1...
 - ProtocolDao...
 - MultiFungibleTokenTemplate1...
 - ERC20PoolTokenTemplate1...
 - UnderwriterPoolDao...
 - LERC20Template1...
 - ShieldPayoutDao...
 - SimplePriceOracleTemplate1...
 - TestPriceFeed...
 - InterestPoolTemplate1...
 - PositionRegistryTemplate1...

============================================================================= test session
starts ==============================================================================
platform darwin -- Python 3.7.4, pytest-5.3.2, py-1.8.1, pluggy-0.13.1
rootdir: /projects/protocol.2.0/lendroid_new/protocol.2.0
plugins: eth-brownie-1.5.1, forked-1.1.3, xdist-1.31.0, web3-5.3.0
collecting 141 items
Launching 'ganache-cli --port 8545 --gasLimit 6721975 --accounts 10 --hardfork istanbul --mnemonic brownie'...
collected 141 items

tests/test_ERC20.py ........
[  5%]
tests/test_ERC20PoolToken.py .......
[ 10%]
tests/test_ERC20TokenPool.py ....
[ 13%]
tests/test_LERC20.py ........
[ 19%]
tests/test_currency_dao.py ..........
[ 26%]
tests/test_interest_pool.py .........
[ 33%]
tests/test_interest_pool_dao.py ..........
[ 40%]
tests/test_market_dao.py ..........
[ 48%]
tests/test_pool_name_registry.py ............
[ 56%]
tests/test_position_registry.py ........
[ 62%]
tests/test_protocol_dao.py ........................
[ 80%]
tests/test_shield_payout_dao.py ........
[ 86%]
tests/test_underwriter_pool.py ......
[ 90%]
tests/test_underwriter_pool_dao.py ............
[100%]

============================================================================== 141 passed in 1085.08s
(0:18:05) ==============================================================================
Terminating local RPC client...
```

Code Coverage

(Note 2020-03-20: this is for commit `eb2ab78`, tests would pass if gas costs were fixed)

The average coverage across all files is currently 65.14%. For some files, the code coverage is less than 50%. It is highly recommended to increase coverage to at least 85% before shipping the code to production.

```
contract: CurrencyDao - 64.9%
  CurrencyDao.initialize - 75.0%
  CurrencyDao._pool_hash - 100.0%
  CurrencyDao._mft_hash - 100.0%
  CurrencyDao._mft_addresses - 100.0%
  CurrencyDao._is_token_supported - 100.0%
  CurrencyDao._deposit_token_to_pool - 87.5%
  CurrencyDao._withdraw_token_from_pool - 75.0%
  CurrencyDao._wrap - 75.0%
  CurrencyDao._unwrap - 75.0%
  CurrencyDao._pause - 100.0%
  CurrencyDao._unpause - 100.0%
  CurrencyDao._transfer_balance_erc20 - 0.0%
  CurrencyDao._transfer_balance_mft - 0.0%
  CurrencyDao.mft_hash - 0.0%
  CurrencyDao.is_token_supported - 100.0%
  CurrencyDao.mft_addresses - 100.0%
  CurrencyDao.f_token - 100.0%
  CurrencyDao.i_token - 100.0%
  CurrencyDao.s_token - 100.0%
  CurrencyDao.u_token - 100.0%
  CurrencyDao.pool_hash - 100.0%
  CurrencyDao.mint_and_self_authorize_erc20 - 75.0%
  CurrencyDao.burn_as_self_authorized_erc20 - 75.0%
  CurrencyDao.set_template - 0.0%
  CurrencyDao.set_token_support - 82.1%
  CurrencyDao.pause - 100.0%
  CurrencyDao.unpause - 100.0%
  CurrencyDao.escape_hatch_erc20 - 0.0%
  CurrencyDao.escape_hatch_mft - 0.0%
  CurrencyDao.wrap - 100.0%
  CurrencyDao.unwrap - 100.0%
```

```
  CurrencyDao.authorized_transfer_l - 75.0%
  CurrencyDao.authorized_transfer_erc20 - 75.0%
  CurrencyDao.authorized_deposit_token - 75.0%
  CurrencyDao.authorized_withdraw_token - 75.0%

contract: ERC20PoolTokenTemplate1 - 81.6%
  ERC20PoolTokenTemplate1.initialize - 75.0%
  ERC20PoolTokenTemplate1.totalSupply - 100.0%
  ERC20PoolTokenTemplate1.allowance - 100.0%
  ERC20PoolTokenTemplate1.transfer - 100.0%
  ERC20PoolTokenTemplate1.transferFrom - 100.0%
  ERC20PoolTokenTemplate1._approve - 100.0%
  ERC20PoolTokenTemplate1.approve - 100.0%
  ERC20PoolTokenTemplate1._mint - 87.5%
  ERC20PoolTokenTemplate1.mint - 50.0%
  ERC20PoolTokenTemplate1.mintAndAuthorizeMinter - 100.0%
  ERC20PoolTokenTemplate1._burn - 100.0%
  ERC20PoolTokenTemplate1.burn - 100.0%
  ERC20PoolTokenTemplate1.burnFrom - 100.0%
  ERC20PoolTokenTemplate1.burnAsAuthorizedMinter - 0.0%

contract: ERC20Template1 - 100.0%
  ERC20Template1.totalSupply - 100.0%
  ERC20Template1.allowance - 100.0%
  ERC20Template1.transfer - 100.0%
  ERC20Template1.transferFrom - 100.0%
  ERC20Template1._approve - 100.0%
  ERC20Template1.approve - 100.0%
  ERC20Template1._mint - 100.0%
  ERC20Template1.mint - 100.0%
  ERC20Template1._burn - 100.0%
  ERC20Template1.burn - 100.0%
  ERC20Template1.burnFrom - 100.0%

contract: ERC20TokenPoolTemplate1 - 85.0%
  ERC20TokenPoolTemplate1.initialize - 83.3%
  ERC20TokenPoolTemplate1.borrowable_amount - 100.0%
  ERC20TokenPoolTemplate1.release - 87.5%
  ERC20TokenPoolTemplate1.destroy - 83.3%

contract: InterestPoolDao - 60.1%
  InterestPoolDao.initialize - 75.0%
  InterestPoolDao._mft_hash - 100.0%
  InterestPoolDao._market_hash - 100.0%
  InterestPoolDao._validate_pool - 75.0%
  InterestPoolDao._LST_stake_value - 87.5%
  InterestPoolDao._stake_LST - 75.0%
  InterestPoolDao._release_staked_LST - 0.0%
  InterestPoolDao.currency_dao - 100.0%
  InterestPoolDao.LST_stake_value - 100.0%
  InterestPoolDao.set_template - 0.0%
  InterestPoolDao.set_minimum_mft_fee - 83.3%
  InterestPoolDao.set_fee_multiplier_per_mft_count - 81.2%
  InterestPoolDao.set_maximum_mft_support_count - 83.3%
  InterestPoolDao._pause - 100.0%
  InterestPoolDao._unpause - 100.0%
  InterestPoolDao.pause - 100.0%
  InterestPoolDao.unpause - 100.0%
  InterestPoolDao._transfer_balance_erc20 - 0.0%
  InterestPoolDao._transfer_balance_mft - 0.0%
  InterestPoolDao.escape_hatch_erc20 - 0.0%
  InterestPoolDao.escape_hatch_mft - 0.0%
  InterestPoolDao.register_pool - 70.2%
  InterestPoolDao.deregister_pool - 0.0%
  InterestPoolDao.register_mft_support - 75.0%
  InterestPoolDao.deregister_mft_support - 0.0%
  InterestPoolDao.deposit_l - 75.0%
  InterestPoolDao._l_to_f_and_i - 75.0%
  InterestPoolDao.split - 100.0%
  InterestPoolDao._i_and_f_to_l - 75.0%
  InterestPoolDao.fuse - 100.0%

contract: InterestPoolTemplate1 - 51.7%
  InterestPoolTemplate1.initialize - 75.0%
  InterestPoolTemplate1._market_hash - 100.0%
  InterestPoolTemplate1._total_pool_share_token_supply - 100.0%
  InterestPoolTemplate1._total_active_contributions - 100.0%
  InterestPoolTemplate1._exchange_rate - 100.0%
  InterestPoolTemplate1._i_token_fee - 50.0%
  InterestPoolTemplate1._estimated_pool_share_tokens - 100.0%
  InterestPoolTemplate1.market_hash - 100.0%
  InterestPoolTemplate1.total_pool_share_token_supply - 100.0%
  InterestPoolTemplate1.l_token_balance - 100.0%
  InterestPoolTemplate1.i_token_balance - 50.0%
  InterestPoolTemplate1.f_token_balance - 100.0%
  InterestPoolTemplate1.total_f_token_balance - 100.0%
  InterestPoolTemplate1.total_active_contributions - 100.0%
  InterestPoolTemplate1.exchange_rate - 100.0%
  InterestPoolTemplate1.estimated_pool_share_tokens - 100.0%
  InterestPoolTemplate1.i_token_fee - 0.0%
  InterestPoolTemplate1.set_public_contribution_acceptance - 0.0%
  InterestPoolTemplate1.set_mft_expiry_limit - 0.0%
  InterestPoolTemplate1.support_mft - 75.0%
  InterestPoolTemplate1.withdraw_mft_support - 0.0%
  InterestPoolTemplate1.set_i_cost_per_day - 75.0%
  InterestPoolTemplate1.decrease_fee_percentage_per_i_token - 0.0%
  InterestPoolTemplate1.withdraw_earnings - 0.0%
  InterestPoolTemplate1.deregister - 0.0%
  InterestPoolTemplate1.increment_i_tokens - 75.0%
  InterestPoolTemplate1.decrement_i_tokens - 90.0%
  InterestPoolTemplate1.exercise_f_tokens - 0.0%
  InterestPoolTemplate1.contribute - 78.1%
  InterestPoolTemplate1.withdraw contribution - 40.8%
```

```
  InterestPoolTemplate1.purchase_i_tokens - 75.0%

contract: LERC20Template1 - 88.3%
  LERC20Template1.initialize - 75.0%
  LERC20Template1.totalSupply - 100.0%
  LERC20Template1.allowance - 100.0%
  LERC20Template1.transfer - 100.0%
  LERC20Template1.transferFrom - 100.0%
  LERC20Template1._approve - 100.0%
  LERC20Template1.approve - 100.0%
  LERC20Template1._mint - 87.5%
  LERC20Template1.mint - 50.0%
  LERC20Template1.mintAndAuthorizeMinter - 100.0%
  LERC20Template1._burn - 100.0%
  LERC20Template1.burn - 100.0%
  LERC20Template1.burnFrom - 100.0%
  LERC20Template1.burnAsAuthorizedMinter - 75.0%

contract: MarketDao - 48.0%
  MarketDao.initialize - 75.0%
  MarketDao._mft_hash - 0.0%
  MarketDao._shield_market_hash - 100.0%
  MarketDao._currency_market_hash - 100.0%
  MarketDao._loan_market_hash - 100.0%
  MarketDao._currency_underlying_pair_hash - 100.0%
  MarketDao._s_payoff - 0.0%
  MarketDao._u_payoff - 0.0%
  MarketDao._currency_remaining_for_auction - 0.0%
  MarketDao._liquidated_underlying_value - 0.0%
  MarketDao.s_payoff - 0.0%
  MarketDao.u_payoff - 0.0%
  MarketDao.currency_remaining_for_auction - 0.0%
  MarketDao.liquidated_underlying_value - 0.0%
  MarketDao._transfer_f_underlying - 75.0%
  MarketDao._open_expiry_market - 75.0%
  MarketDao._open_loan_market - 75.0%
  MarketDao._reset_total_s_payout_value - 0.0%
  MarketDao._settle_loan_market - 0.0%
  MarketDao._open_shield_market - 75.0%
  MarketDao.shield_market_hash - 100.0%
  MarketDao.loan_market_hash - 100.0%
  MarketDao.currency_underlying_pair_hash - 100.0%
  MarketDao.set_template - 0.0%
  MarketDao.set_registry - 87.5%
  MarketDao.set_price_oracle - 80.0%
  MarketDao.maximum_liability_for_currency_market - 100.0%
  MarketDao.set_maximum_liability_for_currency_market - 80.0%
  MarketDao.maximum_liability_for_loan_market - 100.0%
  MarketDao.set_maximum_liability_for_loan_market - 79.2%
  MarketDao.set_auction_slippage_percentage - 83.3%
  MarketDao.set_auction_maximum_discount_percentage - 83.3%
  MarketDao.set_auction_discount_duration - 83.3%
  MarketDao._pause - 100.0%
  MarketDao._unpause - 100.0%
  MarketDao.pause - 100.0%
  MarketDao.unpause - 100.0%
  MarketDao._transfer_balance_erc20 - 0.0%
  MarketDao._transfer_balance_mft - 0.0%
  MarketDao.escape_hatch_auction - 0.0%
  MarketDao.escape_hatch_erc20 - 0.0%
  MarketDao.escape_hatch_mft - 0.0%
  MarketDao.open_shield_market - 75.0%
  MarketDao.settle_loan_market - 0.0%
  MarketDao.process_auction_purchase - 0.0%
  MarketDao.open_position - 75.0%
  MarketDao.close_position - 75.0%
  MarketDao.close_liquidated_position - 0.0%

contract: MultiFungibleTokenTemplate1 - 68.2%
  MultiFungibleTokenTemplate1.initialize - 75.0%
  MultiFungibleTokenTemplate1._hash - 100.0%
  MultiFungibleTokenTemplate1.supportsInterface - 0.0%
  MultiFungibleTokenTemplate1.id - 100.0%
  MultiFungibleTokenTemplate1.is_valid_id - 0.0%
  MultiFungibleTokenTemplate1.setURI - 0.0%
  MultiFungibleTokenTemplate1._create - 70.0%
  MultiFungibleTokenTemplate1.get_or_create_id - 83.3%
  MultiFungibleTokenTemplate1._mint - 100.0%
  MultiFungibleTokenTemplate1.mint - 75.0%
  MultiFungibleTokenTemplate1.burn - 75.0%
  MultiFungibleTokenTemplate1.safeTransferFrom - 75.0%
  MultiFungibleTokenTemplate1.totalBalanceOf - 100.0%
  MultiFungibleTokenTemplate1.balanceOf - 100.0%
  MultiFungibleTokenTemplate1.balanceOfBatch - 0.0%

contract: PoolNameRegistryTemplate1 - 59.1%
  PoolNameRegistryTemplate1.initialize - 75.0%
  PoolNameRegistryTemplate1._name_exists - 100.0%
  PoolNameRegistryTemplate1._lock_name - 75.0%
  PoolNameRegistryTemplate1._unlock_name - 75.0%
  PoolNameRegistryTemplate1._add_name - 90.0%
  PoolNameRegistryTemplate1._remove_name - 87.5%
  PoolNameRegistryTemplate1._pause - 100.0%
  PoolNameRegistryTemplate1._unpause - 100.0%
  PoolNameRegistryTemplate1._transfer_balance_erc20 - 0.0%
  PoolNameRegistryTemplate1.name_exists - 100.0%
  PoolNameRegistryTemplate1.set_name_registration_minimum_stake - 83.3%
  PoolNameRegistryTemplate1.set_name_registration_stake_lookup - 81.2%
  PoolNameRegistryTemplate1.pause - 100.0%
  PoolNameRegistryTemplate1.unpause - 100.0%
  PoolNameRegistryTemplate1.escape_hatch_erc20 - 0.0%
  PoolNameRegistryTemplate1.register_name - 83.3%
  PoolNameRegistryTemplate1.register_name_and_pool - 75.0%
```

```
  PoolNameRegistryTemplate1.register_pool - 0.0%
  PoolNameRegistryTemplate1.deregister_pool - 0.0%
  PoolNameRegistryTemplate1.deregister_name - 75.0%

contract: PositionRegistryTemplate1 - 69.7%
  PositionRegistryTemplate1.initialize - 75.0%
  PositionRegistryTemplate1._loan_market_hash - 0.0%
  PositionRegistryTemplate1._shield_market_hash - 100.0%
  PositionRegistryTemplate1._lock_position - 75.0%
  PositionRegistryTemplate1._unlock_position - 75.0%
  PositionRegistryTemplate1._open_position - 100.0%
  PositionRegistryTemplate1._remove_position - 100.0%
  PositionRegistryTemplate1._partial_or_complete_close_position - 100.0%
  PositionRegistryTemplate1._liquidate_position - 0.0%
  PositionRegistryTemplate1._pause - 100.0%
  PositionRegistryTemplate1._unpause - 100.0%
  PositionRegistryTemplate1.pause - 100.0%
  PositionRegistryTemplate1.unpause - 100.0%
  PositionRegistryTemplate1.avail_loan - 75.0%
  PositionRegistryTemplate1.repay_loan - 75.0%
  PositionRegistryTemplate1.close_liquidated_loan - 0.0%

contract: ProtocolDao - 66.3%
  ProtocolDao._mft_hash - 0.0%
  ProtocolDao._validate_caller - 100.0%
  ProtocolDao.change_governor - 100.0%
  ProtocolDao.change_escape_hatch_manager - 100.0%
  ProtocolDao.change_escape_hatch_token_holder - 100.0%
  ProtocolDao.initialize_pool_name_registry - 75.0%
  ProtocolDao.initialize_position_registry - 75.0%
  ProtocolDao.initialize_currency_dao - 75.0%
  ProtocolDao.initialize_interest_pool_dao - 75.0%
  ProtocolDao.initialize_underwriter_pool_dao - 75.0%
  ProtocolDao.initialize_market_dao - 75.0%
  ProtocolDao.initialize_shield_payout_dao - 75.0%
  ProtocolDao.activate_public_contributions - 100.0%
  ProtocolDao.activate_non_standard_expiries - 100.0%
  ProtocolDao.set_expiry_support - 100.0%
  ProtocolDao.set_registry - 90.0%
  ProtocolDao.set_template - 0.0%
  ProtocolDao.set_pool_name_registration_minimum_stake - 87.5%
  ProtocolDao.set_pool_name_registration_stake_lookup - 91.7%
  ProtocolDao.set_token_support - 87.5%
  ProtocolDao.set_minimum_mft_fee - 91.7%
  ProtocolDao.set_fee_multiplier_per_mft_count - 89.3%
  ProtocolDao.set_maximum_mft_support_count - 91.7%
  ProtocolDao.set_price_oracle - 93.8%
  ProtocolDao.set_maximum_liability_for_currency_market - 93.8%
  ProtocolDao.set_maximum_liability_for_loan_market - 95.0%
  ProtocolDao.set_auction_slippage_percentage - 87.5%
  ProtocolDao.set_auction_maximum_discount_percentage - 87.5%
  ProtocolDao.set_auction_discount_duration - 87.5%
  ProtocolDao.toggle_dao_pause - 83.3%
  ProtocolDao.toggle_registry_pause - 78.1%
  ProtocolDao.escape_hatch_dao_erc20 - 0.0%
  ProtocolDao.escape_hatch_registry_erc20 - 0.0%
  ProtocolDao.escape_hatch_dao_mft - 0.0%
  ProtocolDao.escape_hatch_auction - 0.0%

contract: ShieldPayoutDao - 24.4%
  ShieldPayoutDao.initialize - 75.0%
  ShieldPayoutDao._mft_hash - 0.0%
  ShieldPayoutDao._shield_market_hash - 100.0%
  ShieldPayoutDao._loan_market_hash - 0.0%
  ShieldPayoutDao._s_payoff - 0.0%
  ShieldPayoutDao._u_payoff - 0.0%
  ShieldPayoutDao._settle_s - 0.0%
  ShieldPayoutDao._settle_u - 0.0%
  ShieldPayoutDao.shield_market_hash - 0.0%
  ShieldPayoutDao.s_payoff - 0.0%
  ShieldPayoutDao.u_payoff - 0.0%
  ShieldPayoutDao.register_shield_market - 75.0%
  ShieldPayoutDao._pause - 100.0%
  ShieldPayoutDao._unpause - 100.0%
  ShieldPayoutDao.pause - 100.0%
  ShieldPayoutDao.unpause - 100.0%
  ShieldPayoutDao._transfer_balance_erc20 - 0.0%
  ShieldPayoutDao._transfer_balance_mft - 0.0%
  ShieldPayoutDao.escape_hatch_erc20 - 0.0%
  ShieldPayoutDao.escape_hatch_mft - 0.0%
  ShieldPayoutDao.exercise_s - 0.0%
  ShieldPayoutDao.exercise_u - 0.0%

contract: UnderwriterPoolDao - 61.4%
  UnderwriterPoolDao.initialize - 75.0%
  UnderwriterPoolDao._mft_hash - 100.0%
  UnderwriterPoolDao._market_hash - 100.0%
  UnderwriterPoolDao._validate_pool - 100.0%
  UnderwriterPoolDao._LST_stake_value - 22.5%
  UnderwriterPoolDao._stake_LST - 75.0%
  UnderwriterPoolDao._release_staked_LST - 0.0%
  UnderwriterPoolDao.currency_dao - 100.0%
  UnderwriterPoolDao.LST_stake_value - 100.0%
  UnderwriterPoolDao.set_template - 0.0%
  UnderwriterPoolDao.set_minimum_mft_fee - 83.3%
  UnderwriterPoolDao.set_fee_multiplier_per_mft_count - 81.2%
  UnderwriterPoolDao.set_maximum_mft_support_count - 83.3%
  UnderwriterPoolDao._pause - 100.0%
  UnderwriterPoolDao._unpause - 100.0%
  UnderwriterPoolDao.pause - 100.0%
  UnderwriterPoolDao.unpause - 100.0%
  UnderwriterPoolDao._transfer_balance_erc20 - 0.0%
  UnderwriterPoolDao._transfer_balance_mft - 0.0%
```

```
  UnderwriterPoolDao.escape_hatch_erc20 - 0.0%
  UnderwriterPoolDao.escape_hatch_mft - 0.0%
  UnderwriterPoolDao.register_pool - 70.2%
  UnderwriterPoolDao.deregister_pool - 0.0%
  UnderwriterPoolDao.register_mft_support - 75.0%
  UnderwriterPoolDao.deregister_mft_support - 0.0%
  UnderwriterPoolDao.deposit_l - 87.5%
  UnderwriterPoolDao._l_to_i_and_s_and_u - 75.0%
  UnderwriterPoolDao.split - 95.0%
  UnderwriterPoolDao._i_and_s_and_u_to_l - 75.0%
  UnderwriterPoolDao.fuse - 95.0%

contract: UnderwriterPoolTemplate1 - 48.4%
  UnderwriterPoolTemplate1.initialize - 75.0%
  UnderwriterPoolTemplate1._market_hash - 100.0%
  UnderwriterPoolTemplate1._total_pool_share_token_supply - 100.0%
  UnderwriterPoolTemplate1._total_active_contributions - 100.0%
  UnderwriterPoolTemplate1._exchange_rate - 100.0%
  UnderwriterPoolTemplate1._i_token_fee - 58.3%
  UnderwriterPoolTemplate1._s_token_fee - 58.3%
  UnderwriterPoolTemplate1._estimated_pool_share_tokens - 100.0%
  UnderwriterPoolTemplate1.total_pool_share_token_supply - 100.0%
  UnderwriterPoolTemplate1.l_token_balance - 100.0%
  UnderwriterPoolTemplate1.i_token_balance - 50.0%
  UnderwriterPoolTemplate1.s_token_balance - 100.0%
  UnderwriterPoolTemplate1.u_token_balance - 100.0%
  UnderwriterPoolTemplate1.total_u_token_balance - 100.0%
  UnderwriterPoolTemplate1.total_active_contributions - 100.0%
  UnderwriterPoolTemplate1.exchange_rate - 100.0%
  UnderwriterPoolTemplate1.estimated_pool_share_tokens - 100.0%
  UnderwriterPoolTemplate1.i_token_fee - 0.0%
  UnderwriterPoolTemplate1.s_token_fee - 0.0%
  UnderwriterPoolTemplate1.set_public_contribution_acceptance - 0.0%
  UnderwriterPoolTemplate1.set_mft_expiry_limit - 0.0%
  UnderwriterPoolTemplate1.support_mft - 75.0%
  UnderwriterPoolTemplate1.withdraw_mft_support - 0.0%
  UnderwriterPoolTemplate1.set_i_cost_per_day - 75.0%
  UnderwriterPoolTemplate1.set_s_cost_per_day - 75.0%
  UnderwriterPoolTemplate1.decrease_fee_percentage_per_i_token - 0.0%
  UnderwriterPoolTemplate1.decrease_fee_percentage_per_s_token - 0.0%
  UnderwriterPoolTemplate1.withdraw_earnings - 0.0%
  UnderwriterPoolTemplate1.deregister - 0.0%
  UnderwriterPoolTemplate1.increment_s_tokens - 75.0%
  UnderwriterPoolTemplate1.decrement_s_tokens - 0.0%
  UnderwriterPoolTemplate1.exercise_u_tokens - 0.0%
  UnderwriterPoolTemplate1.contribute - 84.4%
  UnderwriterPoolTemplate1.withdraw_contribution - 36.7%
  UnderwriterPoolTemplate1.purchase_i_tokens - 75.0%
  UnderwriterPoolTemplate1.purchase_s_tokens - 75.0%
```

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
b8125bb9ca52f5b574b2d81472c2240cfb8b24aaafbf1f8ef190e9353ebf352a
./contracts/templates/ERC20PoolTokenTemplate1.vy

f96033bf048fce33db5b8950c52b69867beeb221b8a334692cffcacfeeed9e90 ./contracts/templates/ERC20Template1.vy

f34c9938fb9e602ba64a10f44ddc5960ce3e058ecfd1bc86dd2a1e611e71f9e7
./contracts/templates/ERC20TokenPoolTemplate1.vy

874d5b44420ea67f731e25448f7a089d2631d69a5e52ff3030daa2f957c840eb ./contracts/templates/InterestPoolTemplate1.vy

c5ac91cd2b60e7e156028eb6abb551a31ae2051d8de3b1a165e566f900389a48 ./contracts/templates/LERC20Template1.vy

4dd4182af52bdf8f4ca911014b16dfcccb689d0b8bf6113f8ac87ac3bf2c61f7
./contracts/templates/MultiFungibleTokenTemplate1.vy

9acee63ee8839d357eb7ab4e868cd19be0efc8220d4a4bbcd54a7cc85dada0a8
./contracts/templates/PoolNameRegistryTemplate1.vy

fcf169d24c6e82a2691a4b397e2ec5074cbb371414d8c5377ec9b88d65f45102
./contracts/templates/PositionRegistryTemplate1.vy

47546a2fc7dea2858f8b744419cc7fb392e103dfdc1ebc371701fb1b9321012f
./contracts/templates/SimpleCollateralAuctionCurveTemplate1.vy

43867285e174871e81f88fc218b1debab4a481873b5dcfb510b66780c17ce823
./contracts/templates/SimplePriceOracleTemplate1.vy

db3e5d198e249daaed89e7a6500576d1aa772475ea707ab9289eb70f3b2792aa ./contracts/templates/TestPriceFeed.vy

bf4db52fe74e745ead96a573c072e5d6256771aa2d0649c852447d3a63fb64c3
./contracts/templates/UnderwriterPoolTemplate1.vy

e473cffe003fedabc3105998f6e503c1290f58061f0fd311f5992f54fe7461dc ./contracts/daos/CurrencyDao.vy

ca8f87d32ca36a66ff1c0a82cf7101af956aff8e2fa4dd28948ea860daad2075 ./contracts/daos/InterestPoolDao.vy

c279e2722bd8d1df180c885abffd8b19e0201c7decddeffcca9eff47be510591 ./contracts/daos/MarketDao.vy

e6184a79be80db19f7223bd79771bb9e5be417f17505f374ed9bd3a26aa858d1 ./contracts/daos/ProtocolDao.vy

1172b1dfe3ada18712f177407bc473ad3db400d8f310ee25c0761cf256d6ad58 ./contracts/daos/ShieldPayoutDao.vy

33e0e93547e8447dc2f57b3518f3c57d36c5ab5a11dbc2424c1b223fcf2db551 ./contracts/daos/UnderwriterPoolDao.vy
```

Tests

```
488d71b680544b9bce9435d156c30d51e40e7efa2b1e0139d690a6d315e15c0a ./tests/conftest.py

77b1941d699bcd99265a18a3249f4dd3ff4d7fe4147b2882e9dda679b5c0924a ./tests/test_collateral_auction.py

a38b3f03e646a8cdc84e4f3b5c2a31555e36f0f484827b2c0a62a5235da34680 ./tests/test_currency_dao.py

b4a93639f3105092e14329cb2fb2ce65b61849bd5790771a9d09b48b1ae043f1 ./tests/test_ERC20.py

86e18aa34d80809bc3724758b5a513cb515d736570d3c20c81319110550eed04 ./tests/test_ERC20PoolToken.py

7c158e91c0813aa72f2aff90ed117d145526a1f7705d92f0132f4b517bd408f5 ./tests/test_interest_pool.py

b531d18070f8cab9ebc1b287684e3c43cab14de14a05c1d16cf591443b93060f ./tests/test_interest_pool_dao.py

9a990c305234c227186ad04ede5681ac4ad00081295788a8a2acf7a8802fa61a ./tests/test_LERC20.py

1373607dc53a5cc83135c0411ab56ff7ceffe1c34e77575f8b9c05bb1416d001 ./tests/test_market_dao.py

c98fd12fd46921343529b9c903e505cf30eedf678a2dcd627515eb7fb5ebf1db ./tests/test_pool_name_registry.py

9031fa9992c4371eef2b7f3d330e0cc6c9a00172199d4f59c4a70bf163893c57 ./tests/test_position_registry.py

a5053f8b4ebbef82116bc642817302c9d0e81cdbe7eb05df6f33579da46b11a3 ./tests/test_protocol_dao.py

ecf39bb976ec09815a8b97d6f8ed68c5ead785f1b102b59ac6db9fcc0a47f9be ./tests/test_shield_payout_dao.py

5c95e3c560cab00e718cc5110e37e4877ef5df5adcf006bfe762d5c097c35bfe ./tests/test_underwriter_pool.py

5fb3d7abb9dfe35062dd878c3904818dcb4a27d5e30f8190434fe802ffee8e4a ./tests/test_underwriter_pool_dao.py
```

About Quantstamp

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

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing 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.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.