Quantstamp Security Assessment Certificate

October 20th 2021 — Quantstamp Verified

Reflexer RAI Curve Pool

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

Executive Summary

Type	Automated Market Maker
Auditors	Sebastian Banescu, Senior Research Engineer Hisham Galal, Research Engineer Roman Rohleder, Research Engineer
Timeline	2021-10-08 through 2021-10-19
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Adapting Curve for RAI
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
curve-contract	a6a29eb (audit)
curve-factory	1bf4af5 (audit)
curve-contract	3104128 (reaudit)
curve-factory	854c210 (reaudit)

Total Issues	**10** (1 Resolved)
High Risk Issues	**1** (0 Resolved)
Medium Risk Issues	**1** (0 Resolved)
Low Risk Issues	**2** (0 Resolved)
Informational Risk Issues	**5** (1 Resolved)
Undetermined Risk Issues	**1** (0 Resolved)

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

Summary of Findings

After audit: Quantstamp has performed an audit of the RAI Curve Metapool. The audit was restricted to the following files:

- `StableSwapRAI.vy` in the `curve-contracts` repo
- `DepositRAI.vy` in the `curve-contracts` repo
- `MetaRAI.vy` in the `curve-factory` repo.

During this audit we have identified 10 security issues having various severity levels and 3 deviations from best practices. We recommend addressing all these findings before deploying the smart contracts in production.

After reaudit: Quantstamp has updated the report based on the responses received from the dev team. We note that for all but one Informational severity issue, the dev team has chosen to Acknowledge the findings and keep the code as is due to the fact that addressing the issue could introduce other unforeseen bugs or increase gas costs.

ID	Description	Severity	Status
QSP-1	Memory Corruption Due To Old Compiler Version	High	Acknowledged
QSP-2	Outdated Redemption Price	Medium	Acknowledged
QSP-3	Unlimited Allowance	Low	Acknowledged
QSP-4	Missing Input Validation	Low	Acknowledged
QSP-5	Addresses To Be Updated After Deployment	Informational	Acknowledged
QSP-6	Unlocked Pragma	Informational	Fixed
QSP-7	Variable Shadowing	Informational	Acknowledged
QSP-8	Oracle Dependency	Informational	Acknowledged
QSP-9	Privileged Roles and Ownership	Informational	Acknowledged
QSP-10	Inconsistent Constants Between Code and Curve Specs	Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Mythril 0.22.26
- SmartCheck Released v2.0

Steps taken to run the tools:

Installed the Mythril tool from Pypi: `pip3 install mythril`
Compiled the Vyper sources to EVM bytecode: `vyper StableSwapRAI.vy > StableSwapRAI.bc`
Ran the Mythril tool on each contract: `myth analyze -c StableSwapRAI.bc`
Install SmartCheck globally: `npm install @smartdec/smartcheck -g`
Start the analysis. To start analysis simply run: `smartcheck -p .`

Findings

QSP-1 Memory Corruption Due To Old Compiler Version

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `DepositRAI.vy, StableSwapRAI.vy`

Description: The `DepositRAI.vy` and `StableSwapRAI.vy` files use an old version of the Vyper compiler which is known to contain high severity vulnerabilities such as:

1. Memory corruption when returning a literal struct with a private call inside of it GHSA-xv8x-pr4h-73jv.
2. Storage variables overwritten by re-entrancy locks GHSA-7f92-rr6w-cq64.

Recommendation: We recommend updating the version to the latest compiler version `0.3.0`.

Update: From the dev team:

> We are aware of the issue, but we chose to keep the versions battle tested in production. Both versions used (curve standalone pool and factory implementation) are currently in production and used in several pools storing millions in value. None of them make use of structs and only the factory one uses the `@nonreentrant` decorator but it uses a compiler version (0.2.15) that already includes the patch for this particular issue. Updating both contracts to 0.3.0 would demand additional changes to the code due to breaking changes between major versions, and would further differentiate them from the in production battle tested contracts, increasing the likelihood of issues.

QSP-2 Outdated Redemption Price

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `StableSwapRAI.vy, MetaRAI.vy`

Description: The requirements state that:

> In order to fetch RAI's redemption price, one has to first update it and then retrieve it. This is why the redemption price getter is not a view function. To make it easier to fetch the redemption price, you can use this contract that stores a slightly older version of the price or you can update and fetch the latest price from here.

However, using a slightly older version is a sub-optimal solution that might enable arbitrage opportunities depending on how outdated the prices is.

Recommendation: We recommend first updating the redemption price before retrieving it and hence using either `RedemptionPriceSnap.updateAndGetSnappedPrice()` or `OracleRelayer.redemptionPrice()` as suggested in the requirement quoted above.

Update: From the dev team:

> We are aware of this limitation. Unfortunately the `redemptionPrice()` function on `OracleRelayer` is a state changing function and is expensive to execute, so we chose not to use it and use a snapshot of the price instead, as the gas used to update the variable will outsize the benefit of updating the redemption price in most of the cases. Reasoning is that the `redemptionPrice` is very stable (oscillates no more than 2% around ~US$3) and using a slightly out of date number results in very little deviation. We expect that traders and LPs performing high value operations will be aware of this and will call the update function in the snapshot contract prior to performing these operations.

QSP-3 Unlimited Allowance

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `MetaRAI.vy, DepsoitRAI.vy`

Description: In L178 (MetaRAI) and L88 (DepositRAI), the function `ERC20(coin).approve` is called with the maximum value, which is considered security anti-pattern and could lead to end-user wallet funds being drained if the contract is exploited.

Recommendation: Consider using bounded allowance in the approval to be safe.

Update: From the dev team:

> The pools do perform an unlimited approval to the `base_pool`. We chose to keep them this way because:
>
> - Pools using this pattern have been in production for months, storing millions in value,
> - The contract that holds the approval (`base_pool`) is a well known, trusted contract and will not drain users balances outside normal operations. They are not upgradeable in any way.
> - Changing this will require further changes to the code, that we intentionally tried to keep as close to the original ones as possible.
> - Approving the exact values every time there is a swap/deposit/withdrawal will increase the gas costs of the operations without much benefit.

QSP-4 Missing Input Validation

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `StableSwapRAI.vy, DepositRAI.vy, MetaRAI.vy`

Description: User inputs should always be checked to be in the expected bounds, especially if the inputs come from untrusted users. However, trusted user inputs should also be checked to avoid incidents caused by human error. The following functions do not have proper input validation in place:

1. `_A` , `_fee` and `_admin_fee` in **init**() of `StableSwapRAI.vy` are not validated. Values bigger than `MAX_A`, `MAX_ADMIN_FEE` or `MAX_FEE` can be set here.
2. `_owner`, `_pool_token`, `_base_pool` in `__init__()` of `StableSwapRAI.vy` are not validated. Providing incorrect values may result in a later discovery that the smart contract is not working as intended.
3. `_token` in `__init__()` of `DepositRAI.vy` is not validated. Providing incorrect values may result in a later discovery that the smart contract is not working as intended.
4. On L430, L644, L716, L844, L1051 in `MetaRAI.vy` there is no check that `_receiver` is different from the zero address.

Recommendation: The following recommendations apply to their corresponding point in the description:

1. Adding checks that ensure that `_fee` is less than or equal to `MAX_FEE` and `_admin_fee` is less than or equal to `MAX_ADMIN_FEE`.
2. Adding checks to ensure that all addresses are different from `0x0`.
3. Adding a check to ensure that the address is different from `0x0`.
4. Adding a check to ensure that `_receiver` is different from `0x0`.

Update: From the dev team:

> Here we also chose to follow Curve's implementation. The setup params will be set one and verified by us, the curve team, and the community (there will be governance votes for both adding a gauge in order to grant rewards to the pool and adding it to the frontEnd). Regarding the validation of the receiver, none of the current in production Curve pools validate it. Though we certainly see the benefit in validating it, we feel that the risk of changing the code and the gas cost associated with the verification outweighs it, as the current implementations have not caused any problems up to now (and frontEnd infrastructure will basically be the same).

QSP-5 Addresses To Be Updated After Deployment

Severity: *Informational*

Status: Acknowledged

File(s) affected: `README.md, pooldata.json`

Description: The `README.md` and `pooldata.json` files contain address placeholders for the RAI Curve pool contracts. Users should be able to check the correct address where these contracts were deployed, in order to avoid using the wrong contracts (by mistake or social engineering attacks).

Recommendation: The address placeholders indicated with `0x00...0` must be replaced with the actual deployment addresses once the contracts are deployed.

Update: From the dev team:

> Noted, we will update the contracts after they are deployed.

QSP-6 Unlocked Pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `DepositRAI.vy, StableSwapRAI.vy`

Description: Every Vyper file specifies in the header a version number of the format `@version ^0.2.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked."

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock all the files onto the same Vyper version.

Update: Resolved in PR https://github.com/reflexer-labs/curve-contract/pull/4, by locking `curve-contract/contracts/pools/rai/DepositRAI.vy` and `curve-contract/contracts/pools/rai/StableSwapRAI.vy` to version `0.2.12`

QSP-7 Variable Shadowing

Severity: *Informational*

Status: Acknowledged

File(s) affected: `DepositRAI.vy`

Description: Variable shadowing occurs when a local or `calldata` variable uses the same name as a storage variable. This could lead to ambiguities regarding which variable is actually used.

The following instances have been observed:

1. The local variable `base_pool` inside `__init__` shadows the storage variable `base_pool`.
2. The local variable `base_pool` inside `remove_liquidity_imbalance` shadows the storage variable `base_pool`.
3. The local variable `base_coins` inside `remove_liquidity_imbalance` shadows the storage variable `base_coins`.

Recommendation: Place an underscore in front of local variable names to avoid shadowing.

Update: From the dev team:

> We are aware of the instances outlined in the report. Again we chose to maintain them as they are, as they do not cause unintended behaviour and changing them means further departing from well known and battle tested implementations.

QSP-8 Oracle Dependency

Severity: *Informational*

Status: Acknowledged

File(s) affected: `MetaRAI.vy, StableSwapRAI.vy`

Description: The redemption price data is provided by the external RAI `RedemptionPriceSnap` oracle and is queried in the following functions of `MetaRAI.vy` and `StableSwapRAI.vy`:

- `get_virtual_price()`(, through `_xp()` in the case of `StableSwapRAI.vy`)
- `calc_token_amount()`
- `add_liquidity()`
- `get_dy()`
- `get_dy_underlying()`
- `exchange()`
- `exchange_underlying()`
- `remove_liquidity_imbalance()`
- `calc_withdraw_one_coin()`, through `_calc_withdraw_one_coin()`
- `remove_liquidity_one_coin()`, through `_calc_withdraw_one_coin()`

As this oracle influences the amount of tokens one may get when interacting with the pool its reliance is a potential risk factor if the oracle is attacked.

Recommendation: This risk should be clearly communicated as such in the documentation.

Update: From the dev team:

> There is indeed a dependency on the redemption price, as outlined in QSP-2. The oracle is designed in a very simple fashion, it draws the `redemptionPrice` directly from the RAI system, therefore not allowing any form of manipulation. The risk outlined in QSP-2 still stands, as the `redemptionPrice` in `redemptionPriceSnap` could become outdated in relation to the number in the system, please refer to comment on the aforementioned issue for the reasoning behind it.

QSP-9 Privileged Roles and Ownership

Severity: Informational

Status: Acknowledged

File(s) affected: `MetaRAI.vy, StableSwapRAI.vy`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. In the case of the RAI pool contracts the `Factory(self.factory).admin()` and `self.owner` have privileged roles that may pose a risk to end-users.

Recommendation: 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.

Update: From the dev team:

> The privileged roles of owner and admin on both implementations follow Curve's original design for the pools. These will be granted to governance (either Curve's or Reflexer's) once the pools are fully setup.

QSP-10 Inconsistent Constants Between Code and Curve Specs

Severity: Undetermined

Status: Acknowledged

File(s) affected: `MetaRAI.vy, StableSwapRAI.vy`

Description: The `Curve` interface uses the `BASE_N_COINS` constant with functions `calc_token_amount` (L20) and `add_liquidity` (L26) in `StableSwapRAI.vy`. The same issue occurs in `MetaRAI.vy` on different line numbers. However, the specs show https://curve.readthedocs.io/exchange-deposits.html?highlight=N_COINS#id14 show that `N_COINS` constant is used. Moreover the actual implementation of those two functions also employ the `N_COINS` constant internally as a loop bound. This discrepancy may lead to issues since the values of these 2 constants are different for the RAI pool.

Recommendation: Double check whether `BASE_N_COINS` is the right constant.

Update: From the dev team:

> For both contracts, we followed the implementations currently in use in production. In both instances we used the `MetaUSD` implementation and chose to follow them as they are implemented and in use with no issues noted to date.

Automated Analyses

Mythril

The output of Mythril is indicated below, which we deem to be an informational severity issue:

```
==== Dependence on predictable environment variable ====
SWC ID: 116
Severity: Low
Contract: MAIN
Function name: constructor
PC address: 309
Estimated Gas Usage: 35818 - 193998
A control flow decision is made based on The block.timestamp environment variable.
The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a
malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of
trust into miners.
--------------------
```

SmartCheck

Several warnings of type `VYPER_PRIVATE_MODIFIER_DONT_HIDE_DATA` in `DepositRAI.vy` and `VYPER_PRIVATE_MODIFIER_DONT_HIDE_DATA` in `SwapSwapRAI.vy`) were detected. After checking, these findings are considered as false positives.

Code Documentation

1. The code comment on L157 in `MetaRAI.vy` indicates that: "Addresses of ERC20 conracts of coins". However, this is not accurate.

Adherence to Best Practices

1. There is a typo in the name of the following constant: `REDMPTION_PRICE_SCALE` defined in the `StableSwapRAI.vy` contract.

2. Avoid using long constant values since this could lead to human-error when typing or reading the value. For example, on L106 in `MetaRAI.vy` there is an instance of this issue: `ADMIN_FEE: constant(uint256) = 5000000000`. We recommend using the abbreviated form: `5* 10 ** 9`.

3. The default type of `range` iterators is `int128` and the expected parameter type of `CurveMeta.coins()` is `uint256`. Therefore, there is an implicit cast from `int128` to `uint256` on L66 and L82 in `DepositRAI.vy`. This might lead to underflows in case the value of `i` would be negative. We recommend replacing implicit casts by explicit casts using `convert(i, uint256)`.

Update from the dev team:

> Noted, though we will keep them as they are for the reasons mentioned above. Except for the mentioned typo, the other items were inherited from implementations currently in use, and we feel that changing them for the sake of best practices can bring more risk vs. using the versions battle tested in production.

Test Results

Test Suite Results

We confirm that all 579 tests passed in the `curve-contracts/` repository.

```
============================================================================= test session starts ==============================================================================
platform darwin -- Python 3.9.6, pytest-6.0.1, py-1.10.0, pluggy-0.13.1
rootdir: /curve-contract-rai_pool
plugins: eth-brownie-1.14.5, hypothesis-5.41.3, xdist-1.34.0, web3-5.11.1, forked-1.3.0
collected 579 items

Launching 'ganache-cli --port 8545 --gasLimit 12000000 --accounts 10 --hardfork istanbul --mnemonic brownie --defaultBalanceEther 10000'...
tests/forked/test_gas.py ..                                                                                                                                          [  0%]
tests/forked/test_insufficient_balances.py .                                                                                                                         [  0%]
tests/pools/common/integration/test_heavily_imbalanced.py ..                                                                                                         [  0%]
tests/pools/common/integration/test_virtual_price_increases.py .                                                                                                     [  1%]
tests/pools/common/unitary/test_add_liquidity.py .........                                                                                                           [  2%]
tests/pools/common/unitary/test_add_liquidity_initial.py ....                                                                                                        [  3%]
tests/pools/common/unitary/test_claim_fees.py ......                                                                                                                 [  4%]
tests/pools/common/unitary/test_exchange.py ......................                                                                                                   [  8%]
tests/pools/common/unitary/test_exchange_reverts.py ..............                                                                                                   [ 10%]
tests/pools/common/unitary/test_exchange_underlying.py ..................................................................................................................... [ 32%]
.................................................                                                                                                                    [ 41%]
tests/pools/common/unitary/test_exchange_underlying_reverts.py .....................................                                                                 [ 48%]
tests/pools/common/unitary/test_get_virtual_price.py ...................                                                                                             [ 51%]
tests/pools/common/unitary/test_kill.py ...........                                                                                                                  [ 53%]
tests/pools/common/unitary/test_modify_fees.py ......................                                                                                                [ 57%]
tests/pools/common/unitary/test_nonpayable.py .                                                                                                                      [ 57%]
tests/pools/common/unitary/test_ramp_A_precise.py ........                                                                                                           [ 59%]
tests/pools/common/unitary/test_remove_liquidity.py .......                                                                                                          [ 60%]
tests/pools/common/unitary/test_remove_liquidity_imbalance.py ............                                                                                           [ 62%]
tests/pools/common/unitary/test_remove_liquidity_one_coin.py ..........................                                                                              [ 67%]
tests/pools/common/unitary/test_transfer_ownership.py ..........                                                                                                     [ 68%]
tests/pools/common/unitary/test_xfer_to_contract.py ..                                                                                                               [ 69%]
tests/pools/meta/test_exchange_with_rate.py ..                                                                                                                       [ 69%]
tests/pools/meta/test_rate_caching.py ...................                                                                                                            [ 72%]
tests/pools/meta/integration/test_rate_handling.py .                                                                                                                 [ 72%]
tests/pools/rai/integration/test_redemption_rate_handling.py .                                                                                                       [ 73%]
tests/pools/rai/unitary/test_add_liquidity_initial_moving_rp.py ........                                                                                             [ 74%]
tests/pools/rai/unitary/test_add_liquidity_moving_rp.py ....                                                                                                         [ 75%]
tests/pools/rai/unitary/test_exchange_moving_rp.py ...                                                                                                               [ 75%]
tests/pools/rai/unitary/test_exchange_underlying_moving_rp.py ..............................................                                                         [ 83%]
tests/pools/rai/unitary/test_remove_liquidity_imbalance_moving_rp.py ..................                                                                              [ 87%]
tests/pools/rai/unitary/test_remove_liquidity_moving_rp.py ..                                                                                                        [ 87%]
tests/pools/rai/unitary/test_rp_caching.py .                                                                                                                         [ 87%]
tests/zaps/common/test_add_liquidity_initial_zap.py .....                                                                                                            [ 88%]
tests/zaps/common/test_add_liquidity_zap.py .............                                                                                                            [ 90%]
tests/zaps/common/test_remove_liquidity_imbalance_zap.py ............................                                                                                [ 95%]
tests/zaps/common/test_remove_liquidity_one_coin_zap.py ........                                                                                                     [ 97%]
tests/zaps/common/test_remove_liquidity_zap.py ............                                                                                                          [ 99%]
tests/zaps/common/test_return_values.py ...                                                                                                                          [ 99%]
tests/zaps/meta/integration/test_remove_liquidity_imbalance_zap_meta.py .                                                                                            [100%]
============================================================================== warnings summary ===============================================================================
tests/pools/common/integration/test_virtual_price_increases.py:8
  /curve-contract-rai_pool/tests/pools/common/integration/test_virtual_price_increases.py:8: PytestUnknownMarkWarning: Unknown pytest.mark.skip_meta - is this a typo?  You can register custom marks to avoid this warning -
for details, see https://docs.pytest.org/en/stable/mark.html
    pytest.mark.skip_meta,
-- Docs: https://docs.pytest.org/en/stable/warnings.html
================================================================== 579 passed, 1 warning in 1278.85s (0:21:18) ==================================================================
```

Code Coverage

We were not able to obtain any coverage data from the test suite because of technical limitations. However, we note that the contracts have been fuzz-tested.

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

```
3057075b16c6103d993ca50b4d2e218b03e3898a1ec5029794dbd7fe3d6423bd ./DepositRAI.vy

1f773238c73cbce3a23ca5472d48c9d44d8894a0e20b2bee6415e786be126211 ./StableSwapRAI.vy

f5e1f89a039df85f87e8f6f45991fd48364850de0f7ecea457d3d16d1f01af75 ./meta/MetaRAI.vy
```

Tests

```
b95a4dce6d49027bce7af2286233b552c55f120000f4d07080d18b1e39b7ce3f ./rai/integration/test_redemption_rate_handling.py

60339813069aa433f4abf135a0ee1c8b1a933dbb3b96720edc79180618f4adc6 ./rai/unitary/test_rp_caching.py

c954e4dfd7e56770fc67a094b0e1354a8e3254b34c0758021095d73ac63429d9 ./rai/unitary/test_add_liquidity_initial_moving_rp.py

e83c7d902d870f6b860a0a9c36529ed6c43aad7e70e4b175ccdf2b6026b302c4 ./rai/unitary/test_remove_liquidity_imbalance_moving_rp.py

95884c7445c0c9507db322c24bf26bf2de7008e3e5431c159054b78fd949706b ./rai/unitary/test_remove_liquidity_moving_rp.py

0f7a49c7754f0ca9db29d03a22809a36a97cbb9e63495d90d35ca1e6c26faecc ./rai/unitary/test_exchange_moving_rp.py

7b9327b5366eb1a0654ac79444f2763a10a8b4f714742fb8bcb93804a2c083b5 ./rai/unitary/test_exchange_underlying_moving_rp.py

b564057db39fb51aca62e7aa81c77e01f940a710948edc77b63e6d603893a422 ./rai/unitary/test_add_liquidity_moving_rp.py
```

Changelog

- 2021-10-15 - Initial 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.