Quantstamp Contract Security Certificate

October 15th 2019 — Quantstamp Verified

Ampleforth

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

Executive Summary

Type	Digital asset protocol
Auditors	Kacper Bąk, Senior Research Engineer Martin Derka, Senior Research Engineer Sung-Shine Lee, Research Engineer
Timeline	2019-05-13 through 2019-06-12
EVM	Byzantium
Languages	Javascript, Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Red Book, The Ampleforth Protocol Ampleforth - A New Synthetic Commodity

Source Code

Repository	Commit
uFragments	1ca2ae2
market-oracle	8bbe43c

Total Issues	**8** (6 Resolved)
High Risk Issues	0
Medium Risk Issues	0
Low Risk Issues	6 (4 Resolved)
Informational Risk Issues	2 (2 Resolved)
Undetermined Risk Issues	0

Overall Assessment

The code is well-written, well-documented, and well-tested. We have not found any significant security vulnerabilities, but a few low risk issues that are the result of certain design choices. We classified the issues as low risk since they are unlikely to occur and have low impact.

Severity Categories

High	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	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	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.

Goals

- Is the protocol numerically stabile?
- Are contract upgrades implemented correctly?
- Are there any issues with oracles security?

Changelog

- 2019-05-17 - Initial Report
- 2019-06-12 - Revised Report based on commit `1ca2ae2`
- 2019-06-13 - Revised Report based on commit `055eb16`

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

Setup

Tool Setup:

- Truffle v4.1.14
- Oyente v1.2.5
- Mythril v0.2.7
- MAIAN commit sha: ab387e1
- Securify

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
3. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
4. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
5. Installed the Mythril tool from Pypi: `pip3 install mythril`
6. Ran the Mythril tool on each contract: `myth -x path/to/contract`
7. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
8. Installed the Oyente tool from Docker: `docker pull luongnguyen/oyente`
9. Migrated files into Oyente (root directory): `docker run -v $(pwd):/tmp - it luongnguyen/oyente`
10. Ran the Oyente tool on each contract: `cd /oyente/oyente && python oyente.py /tmp/path/to/contract`
11. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
12. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`

Assessment

Findings

gon **balances may become untransferable over time**

Severity: *Low*

Contract(s) affected: `UFragments.sol`

Description: The functions `transfer()` and `transferFrom()` take `value` that signifies the number of tokens to transfer. Both functions compute the corresponding number of *gons* and use them to update the ledger. It is possible that the value of `_gonsPerFragment` becomes large enough (due to constantly increasing target rate) so that `gonValue` exceeds the account balance even if `value` equals 1. Such a transfer will fail on `SafeMath.sub()`. Consequently, the source account will have a non-zero *gon* balance that will be untransferable. Theoretically, a large number of *gons* can be dispersed among multiple accounts in too low quantities, so all those *gons* will become unspendable.

Exploit Scenario:

1. User has 1 token corresponding to N *gons* at the beginning of the token lifecycle.
2. The target rate of the token increases over time.
3. When the target rate increases, each `supplyDelta` passed to `rebase()` is negative, which results in decreased `_totalSupply` of tokens.
4. Consequently, `_gonsPerFragment` increases and so does `gonValue` used in `transfer()`. It means that over time for a constant `value` more *gons* are required to perform the transfer.
5. User has N *gons* and tries to transfer 1 token whose value is now larger than the N *gons*.
6. The transfer fails and the tokens become unspendable.

Recommendation: Add a method for transferring *gons* between accounts.

Ampleforth response: this is a known behavior, but has very minimal impact. Let's derive specific bounds.

AMPL balances can be denominated down to 9 decimal places, so the maximum untransferable *gons* per wallet has a value of less than `TargetPrice` * 10^(-9).

If you consider a `TargetPrice` = $1 and 50M wallets, the maximum total untransferable *gons* for the whole network is $1 * 10^(-9) * 50M, or $0.05.

Centralization of Power

Severity: *Low*

Status: Fixed

Contract(s) affected: `UFragments.sol`, `UFragmentsPolicy.sol`, `MedianOracle.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

The owner of `UFragments` may (un)pause rebasing and transfers.

The owner of `UFragmentsPolicy` may set oracles and policy parameters.

The owner of `MedianOracle` may arbitrarily whitelist data providers.

Recommendation: This centralization of power needs to be made clear to the users. The level of privilege the contracts allow to the owner makes Ampleforth, essentially, a centralized token. We recommend, as much as possible, removing the extra privileges from the owner.

Ampleforth response: it is a concern around governance and access power. Note that the owner can't make changes that affect wallets, shares of the network, or any changes pertaining to individual accounts.

The `owner` variable referenced in the contract code identifies a particular Ethereum address that has special abilities. In the beginning this address will be controlled by the Ampleforth development team via a Gnosis 2-of-4 multisig wallet contract. However, this will eventually be the address of the onchain governance module.

Using a single address variable allows us to separate the concerns of governance from the concerns of the main protocol logic. For example, this interface lets the governance evolve from a multisig M-of-N wallet to a fully binding onchain voting mechanism without having to make any changes to the protocol code itself.

Blog post published 6/10

Token may fail to converge to the target rate if `rebase()` keeps failing

Severity: *Low*

Status: Fixed

Contract(s) affected: `UFragmentsPolicy.sol`, `UFragments.sol`

Description: The function `getData()` iterates over all reports from the providers and counts the number of valid reports (i.e., not too old and not too recent). `rebase()` uses the return value from `getData()` and reverts if there are no valid reports. In case of a revert, no information is logged by `getData()` and the token fails to apply the previously computed dampening factor. If `rebase()` keeps failing over the course of `rebaseLag` (days), then the price will never reach the previously computed target rate.

Recommendation: We recommend updating `rebase()` so that over the course of `rebaseLag` (days) the dampening factor can still be applied if there are no valid reports.

Ampleforth response: the policy does not store historical exchange rates or supply adjustments. Each day's supply adjustment is based on the previous 24hr market price provided by the oracle, smoothed out with a dampening factor of 1/30. There is no carryover from previous computations. With lack of valid market data, the policy has nothing to base supply adjustment on.

Gas Usage / `for` Loop Concerns

Severity: *Low*

Status: Fixed

Contract(s) affected: `Select.sol`, `MedianOracle.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to finish, then it may prevent the contract from functioning correctly entirely. Specifically, the gas usage in `for` loops depends on the number of providers and reports.

Recommendation: Although the problem is unlikely to occur in case of *few* iterations, we recommend adding functions that allow to break up single `for` loops into multiple transactions (e.g., by specifying the start index and the number of iterations). In some scenarios, `for` loops and arrays can be replaced by LinkedListLib from Modular.

Ampleforth response: given that these for-loops are time dependent, breaking up the loop across transactions is not a good option for our use case.

The current design can handle more than 120 providers with valid reports per Oracle before hitting the block gas limit. In the median aggregation function, we use an insertion sort because (based on our measurements) it is more efficient on small numbers of providers like we'll have in the beginning. If we ever reach that too many providers, we can switch to a more efficient selection algorithm - e.g., quickselect - or switch to offchain aggregation w/ signatures like MakerDAO's structure. However, we prefer the transparency of onchain aggregation as much as possible.

Malicious providers may perform DoS by making `Select.computeMedian()` revert

Severity: *Low*

Status: Fixed

Contract(s) affected: `MedianOracle.sol`, `Select.sol`

Description: `getData()` (used by `rebase()`) uses the function `computeMedian()` to compute median of reported values from providers. In case of an even number of providers, it adds two middle values from the array. Malicious providers could collude and provide values that cause an overflow and, consequently, revert `rebase()`.

Recommendation: Although providers are trusted entities and could be manually removed from the whitelist, we recommend off-chain monitoring of the values supplied by the providers to detect any potential issues as early as possible.

Ampleforth response: for a malicious data provider to cause `rebase()` to revert by overflowing the addition, they'd have to report a rate which is one of the middle two values. For such a large value to be in the middle, they'd have to either collude with or compromise 50% of the other providers. The purpose of using the median function for aggregation is to reduce the dependence on any single provider for exactly situations like this. As per the recommendation we do have monitoring and alerting services setup to monitor the validity and correctness of the data returned by the oracles, individual data providers and all on-chain hyper parameters. We are also working on (1) increasing the number of providers after launch and (2) a publicly viewable page on the dashboard that provides full transparency and auditability of the values provided by providers.

Block Timestamp Manipulation

Severity: *Low*

Contract(s) affected: `MedianOracle.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account.

Specifically, `reportDelaySec` is used to filter out too recent reports supplied by providers. Similarly, `reportExpirationTimeSec` is used to filter out too old reports. Both variables may take arbitrary small values, although the whitepaper specifies that a market report must exist on-chain publicly for at least 1 hour before it can be used by the supply policy and that a market report will expire on-chain if a new report is not provided before 6 hours elapses. Consequently, for sufficiently small values, a miner might be able to manipulate the block timestamp so that a specific report is taken into account (or not) by `rebase()`.

Recommendation: As long as it is practical, we recommend using block numbers instead of seconds to measure time. Alternatively, specify minimum reasonable values that each variable may take so that slight timestamp manipulation is benign.

Ampleforth response: block numbers are not accurate enough for time period representation for our use cases, having predictable time behavior for rebase is important for marketplace actors.

As these variables are only modifiable by contract owner and in the future through governance, We choose to not enforce minimum values for these variables until the addition of onchain governance.

Race Conditions / Front-Running

Severity: *Informational*

Status: Fixed

Contract(s) affected: `UFragmentsPolicy.sol`, `UFragments.sol`

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. A miner can take advantage of this by generating and moving transactions in a way that benefits themselves.

In `UFragmentsPolicy`, there is a transaction ordering dependency between the functions `setDeviationThreshold()` (which sets the deviation threshold) and `rebase()` (which adjusts the token supply).

In `UFragments`, there is a transaction ordering dependency between the functions `transfer()` (which sets the deviation threshold) and `rebase()` (which adjusts `_gonsPerFragment`).

A miner could take advantage of the ordering of transactions to sell their tokens at a higher rate than others.

Exploit Scenario:

1. User calls `UFragmentsPolicy.rebase()` to adjust the supply and, indirectly, `_gonsPerFragment` so that its value increases.
2. Miner notices the transaction and realizes that due to increased `_gonsPerFragment`, they'll have less tokens.
3. The miner submits `transfer()` (e.g., as a part of an atomic swap with another token) to sell tokens, and orders it so that it gets executed before `rebase()`.

Recommendation: None.

Ampleforth response: the scheduling of rebase operations is fully public, so there's already widespread understanding of when they will take place. Additionally, the oracle rates are 24hr VWAP and the reports must also exist onchain for at least 1 hour, so the magnitude of rebase adjustments are also public and known ahead of time.

So while the miner who releases the next block knows which specific block number the rebase operation will get mined in, it doesn't provide meaningful advantage over users who could also use the same arbitrage opportunity many blocks beforehand. We welcome any arbitrage opportunity that's based on public information, because it encourages efficient price discovery in the market.

Allowance Double-Spend Exploit

Severity: *Informational*

Status: Fixed

Contract(s) affected: `UFragments.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario:

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 (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and `decreaseAllowance` which are already present in the code and address the double-spend issue.

Test Results

Test Suite Results

All tests executed successfully.

```
Contract: SafeMathInt
  add
    ✓ adds correctly (109ms)
    ✓ should fail on addition overflow (170ms)
    ✓ should fail on addition overflow, swapped args (97ms)
    ✓ should fail on addition negative overflow (78ms)
  sub
    ✓ subtracts correctly (63ms)
    ✓ should fail on subtraction overflow (38ms)
    ✓ should fail on subtraction negative overflow (44ms)
  mul
    ✓ multiplies correctly (57ms)
    ✓ handles a zero product correctly (61ms)
    ✓ should fail on multiplication overflow (52ms)
    ✓ should fail on multiplication negative overflow (51ms)
    ✓ should fail on multiplication between -1 and MIN_INT256 (51ms)
  div
    ✓ divides correctly
    ✓ should fail on zero division (94ms)
    ✓ should fail when MIN_INT256 is divided by -1
  abs
    ✓ works for 0 (42ms)
    ✓ works on positive numbers
    ✓ works on negative numbers
    ✓ fails on overflow condition

Contract: UFragments
   ✓ should reject any ether sent to it

Contract: UFragments:Initialization
   ✓ should transfer 50M uFragments to the deployer
   ✓ should set the totalSupply to 50M
   ✓ should set the owner
   ✓ should set detailed ERC20 parameters (71ms)
   ✓ should have 9 decimals
   ✓ should have AMPL symbol (43ms)

Contract: UFragments:setMonetaryPolicy
   ✓ should set reference to policy contract (69ms)
   ✓ should emit policy updated event (46ms)

Contract: UFragments:setMonetaryPolicy:accessControl
   ✓ should be callable by owner

Contract: UFragments:setMonetaryPolicy:accessControl
   ✓ should NOT be callable by non-owner

Contract: UFragments:PauseRebase
   ✓ should emit pause event
   ✓ should not allow calling rebase
   ✓ should allow calling transfer (59ms)
   ✓ should allow calling approve (59ms)
   ✓ should allow calling allowance
   ✓ should allow calling transferFrom (52ms)
   ✓ should allow calling increaseAllowance (39ms)
   ✓ should allow calling decreaseAllowance (46ms)
   ✓ should allow calling balanceOf
   ✓ should allow calling totalSupply

Contract: UFragments:PauseRebase:accessControl
   ✓ should be callable by owner (43ms)
   ✓ should NOT be callable by non-owner

Contract: UFragments:PauseToken
   ✓ should emit pause event
   ✓ should allow calling rebase (50ms)
   ✓ should not allow calling transfer
   ✓ should not allow calling approve
   ✓ should allow calling allowance
   ✓ should not allow calling transferFrom
   ✓ should not allow calling increaseAllowance
   ✓ should not allow calling decreaseAllowance (47ms)
   ✓ should allow calling balanceOf
   ✓ should allow calling totalSupply

Contract: UFragments:PauseToken:accessControl
   ✓ should be callable by owner (55ms)
   ✓ should NOT be callable by non-owner

Contract: UFragments:Rebase:accessControl
   ✓ should be callable by monetary policy (42ms)
   ✓ should not be callable by others

Contract: UFragments:Rebase:Expansion
   ✓ should increase the totalSupply
   ✓ should increase individual balances
   ✓ should emit Rebase
   ✓ should return the new supply (135ms)

Contract: UFragments:Rebase:Expansion
  when totalSupply is less than MAX_SUPPLY and expands beyond
    ✓ should increase the totalSupply to MAX_SUPPLY
    ✓ should emit Rebase
  when totalSupply is MAX_SUPPLY and expands
    ✓ should NOT change the totalSupply
    ✓ should emit Rebase

Contract: UFragments:Rebase:NoChange
   ✓ should NOT CHANGE the totalSupply
   ✓ should NOT CHANGE individual balances
   ✓ should emit Rebase

Contract: UFragments:Rebase:Contraction
   ✓ should decrease the totalSupply
   ✓ should decrease individual balances
   ✓ should emit Rebase

Contract: UFragments:Transfer
  deployer transfers 12 to A
    ✓ should have correct balances (72ms)
  deployer transfers 15 to B
    ✓ should have balances [973,15] (96ms)
  deployer transfers the rest to C
    ✓ should have balances [0,973] (77ms)
  when the recipient address is the contract address
    ✓ reverts on transfer
    ✓ reverts on transferFrom
  when the recipient is the zero address
    ✓ emits an approval event
    ✓ transferFrom should fail (39ms)

Contract: UFragmentsPolicy
   ✓ should reject any ether sent to it

Contract: UFragmentsPolicy:initialize
  initial values set correctly
    ✓ deviationThreshold
    ✓ rebaseLag
    ✓ minRebaseTimeIntervalSec
    ✓ epoch
    ✓ rebaseWindowOffsetSec
    ✓ rebaseWindowLengthSec
    ✓ should set owner
    ✓ should set reference to uFragments

Contract: UFragmentsPolicy:setMarketOracle
   ✓ should set marketOracle

Contract: UFragments:setMarketOracle:accessControl
   ✓ should be callable by owner
   ✓ should NOT be callable by non-owner

Contract: UFragmentsPolicy:setCpiOracle
   ✓ should set cpiOracle (38ms)

Contract: UFragments:setCpiOracle:accessControl
   ✓ should be callable by owner
   ✓ should NOT be callable by non-owner

Contract: UFragmentsPolicy:setDeviationThreshold
   ✓ should set deviationThreshold

Contract: UFragments:setDeviationThreshold:accessControl
   ✓ should be callable by owner
   ✓ should NOT be callable by non-owner

Contract: UFragmentsPolicy:setRebaseLag
  when rebaseLag is more than 0
    ✓ should setRebaseLag (40ms)
  when rebaseLag is 0
    ✓ should fail

Contract: UFragments:setRebaseLag:accessControl
   ✓ should be callable by owner
   ✓ should NOT be callable by non-owner

Contract: UFragmentsPolicy:setRebaseTimingParameters
  when interval=0
    ✓ should fail
  when offset > interval
    ✓ should fail
  when params are valid
    ✓ should setRebaseTimingParameters (68ms)

Contract: UFragments:setRebaseTimingParameters:accessControl
   ✓ should be callable by owner
   ✓ should NOT be callable by non-owner
```

```
  Contract: UFragmentsPolicy:Rebase
    when minRebaseTimeIntervalSec has NOT passed since the previous rebase
      ✓ should fail

  Contract: UFragmentsPolicy:Rebase
    when rate is within deviationThreshold
      ✓ should return 0 (771ms)

  Contract: UFragmentsPolicy:Rebase
    when rate is more than MAX_RATE
      ✓ should return same supply delta as delta for MAX_RATE (737ms)

  Contract: UFragmentsPolicy:Rebase
    when uFragments grows beyond MAX_SUPPLY
      ✓ should apply SupplyAdjustment {MAX_SUPPLY - totalSupply} (87ms)

  Contract: UFragmentsPolicy:Rebase
    when uFragments supply equals MAX_SUPPLY and rebase attempts to grow
      ✓ should not grow (69ms)

  Contract: UFragmentsPolicy:Rebase
    when the market oracle returns invalid data
      ✓ should fail (170ms)
    when the market oracle returns valid data
      ✓ should NOT fail (203ms)

  Contract: UFragmentsPolicy:Rebase
    when the cpi oracle returns invalid data
      ✓ should fail (170ms)
    when the cpi oracle returns valid data
      ✓ should NOT fail (196ms)

  Contract: UFragmentsPolicy:Rebase
    positive rate and no change CPI
      ✓ should increment epoch
      ✓ should update lastRebaseTimestamp
      ✓ should emit Rebase with positive requestedSupplyAdjustment
      ✓ should call getData from the market oracle
      ✓ should call getData from the cpi oracle
      ✓ should call uFrag Rebase

  Contract: UFragmentsPolicy:Rebase
    negative rate
      ✓ should emit Rebase with negative requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    when cpi increases
      ✓ should emit Rebase with negative requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    when cpi decreases
      ✓ should emit Rebase with positive requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    rate=TARGET_RATE
      ✓ should emit Rebase with 0 requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    when its 5s after the rebase window closes
      ✓ should fail (149ms)
    when its 5s before the rebase window opens
      ✓ should fail (148ms)
    when its 5s after the rebase window opens
      ✓ should NOT fail (208ms)
    when its 5s before the rebase window closes
      ✓ should NOT fail (214ms)

  Contract: UInt256Lib
    toInt256Safe
      when then number is more than MAX_INT256
        ✓ should fail
      when then number is MAX_INT256
        ✓ converts int to uint256 safely
      when then number is less than MAX_INT256
        ✓ converts int to uint256 safely
      when then number is 0
        ✓ converts int to uint256 safely

  Contract: UFragments:ERC20
    totalSupply
      ✓ returns the total amount of tokens
    balanceOf
      when the requested account has no tokens
        ✓ returns zero
      when the requested account has some tokens
        ✓ returns the total amount of tokens

  Contract: UFragments:ERC20:transfer
    when the sender does NOT have enough balance
      ✓ reverts (43ms)
    when the sender has enough balance
      ✓ should transfer the requested amount (57ms)
      ✓ should emit a transfer event
    when the recipient is the zero address
      ✓ should fail

  Contract: UFragments:ERC20:transferFrom
    when the spender does NOT have enough approved balance
      when the owner does NOT have enough balance
        ✓ reverts (49ms)
      when the owner has enough balance
        ✓ reverts (49ms)
    when the spender has enough approved balance
      when the owner does NOT have enough balance
        ✓ should fail (57ms)
      when the owner has enough balance
        ✓ transfers the requested amount
        ✓ decreases the spender allowance
        ✓ emits a transfer event

  Contract: UFragments:ERC20:approve
    when the spender is NOT the zero address
      when the sender has enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ approves the requested amount and replaces the previous one
          ✓ emits an approval event
      when the sender does not have enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ approves the requested amount
          ✓ emits an approval event

  Contract: UFragments:ERC20:increaseAllowance
    when the spender is NOT the zero address
      when the sender has enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ increases the spender allowance adding the requested amount
          ✓ emits an approval event
      when the sender does not have enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ increases the spender allowance adding the requested amount
          ✓ emits an approval event

  Contract: UFragments:ERC20:decreaseAllowance
    when the spender is NOT the zero address
      when the sender does NOT have enough balance
        when there was no approved amount before
          ✓ keeps the allowance to zero
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ decreases the spender allowance subtracting the requested amount
          ✓ emits an approval event
      when the sender has enough balance
        when there was no approved amount before
          ✓ keeps the allowance to zero
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ decreases the spender allowance subtracting the requested amount
          ✓ emits an approval event


  168 passing (30s)



  Contract: MedianOracle:GasTests
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78319
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63329
    when the sources are live
getData() gas: 223817
      ✓ should calculate the combined market rate and volume (313ms)

  Contract: MedianOracle:constructor
    ✓ should fail if a parameter is invalid (264ms)

  Contract: MedianOracle:providersSize
    ✓ should return the number of sources added to the whitelist (182ms)

  Contract: MedianOracle:addProvider
    when successful
      ✓ should emit SourceAdded message
      ✓ should add source to the whitelist
      ✓ should not add an existing source to the whitelist

  Contract: MedianOracle:pushReport
    ✓ should only push from authorized source (156ms)
    ✓ should emit ProviderReportPushed message (143ms)

  Contract: MedianOracle:addProvider:accessControl
    ✓ should be callable by owner (48ms)
    ✓ should NOT be callable by non-owner

  Contract: MedianOracle:removeProvider
    when source is part of the whitelist
      ✓ should emit SourceRemoved message
      ✓ should remove source from the whitelist (110ms)

  Contract: MedianOracle:removeProvider
    ✓ Remove last element (138ms)
    ✓ Remove middle element (129ms)
    ✓ Remove only element (257ms)

  Contract: MedianOracle:removeProvider
    ✓ when source is NOT part of the whitelist (287ms)

  Contract: MedianOracle:removeProvider:accessControl
    ✓ should be callable by owner (49ms)
    ✓ should NOT be callable by non-owner

  Contract: MedianOracle:getData
    when the reports are valid
      ✓ should calculate the combined market rate and volume (128ms)

  Contract: MedianOracle:getData
    when one of reports has expired
      ✓ should emit ReportTimestampOutOfRange message (129ms)
      ✓ should calculate the exchange rate (138ms)

  Contract: MedianOracle:getData
    when one of the reports is too recent
      ✓ should emit ReportTimestampOutOfRange message (118ms)
      ✓ should calculate the exchange rate (112ms)

  Contract: MedianOracle:getData
    when not enough providers are valid
      ✓ should emit ReportTimestampOutOfRange message (97ms)
      ✓ should not have a valid result (119ms)

  Contract: MedianOracle:getData
    when all reports have expired
      ✓ should emit 2 ReportTimestampOutOfRange messages (71ms)
      ✓ should return false and 0 (62ms)

  Contract: MedianOracle:getData
    when recent is too recent and past is too old
      ✓ should emit ReportTimestampOutOfRange message (66ms)
      ✓ should fail (51ms)

  Contract: MedianOracle:getData
    when recent is too recent and past is too recent
      ✓ should emit ReportTimestampOutOfRange message (80ms)
      ✓ should fail (63ms)

  Contract: MedianOracle:getData
    when recent is too recent and past is valid
      ✓ should succeeded (54ms)

  Contract: MedianOracle:getData
    when recent is not too recent nor too old
      ✓ should succeed (54ms)

  Contract: MedianOracle:getData
    when recent is not too recent but too old
      ✓ should fail (46ms)

  Contract: MedianOracle:PurgeReports
    ✓ data not available after purge (51ms)
    ✓ data available after another report (108ms)
    ✓ cannot purge a non-whitelisted provider (134ms)

  Contract: Select
    Select:computeMedian
      ✓ median of 1 (46ms)
      ✓ median of 2 (47ms)
      ✓ median of 3 (54ms)
      ✓ median of odd sized list (364ms)
      ✓ median of even sized list (495ms)
      ✓ not enough elements in array
      ✓ median of empty list
      ✓ median of list of size 0


  45 passing (16s)
```

Code Coverage

The code features excellent code coverage.

```
  Contract: SafeMathInt
    add
      ✓ adds correctly (109ms)
      ✓ should fail on addition overflow (170ms)
      ✓ should fail on addition overflow, swapped args (97ms)
      ✓ should fail on addition negative overflow (78ms)
    sub
      ✓ subtracts correctly (63ms)
      ✓ should fail on subtraction overflow (38ms)
      ✓ should fail on subtraction negative overflow (44ms)
    mul
      ✓ multiplies correctly (57ms)
      ✓ handles a zero product correctly (61ms)
      ✓ should fail on multiplication overflow (52ms)
      ✓ should fail on multiplication negative overflow (51ms)
      ✓ should fail on multiplication between -1 and MIN_INT256 (51ms)
    div
      ✓ divides correctly
      ✓ should fail on zero division (94ms)
      ✓ should fail when MIN_INT256 is divided by -1
    abs
      ✓ works for 0 (42ms)
      ✓ works on positive numbers
      ✓ works on negative numbers
      ✓ fails on overflow condition

  Contract: UFragments
    ✓ should reject any ether sent to it

  Contract: UFragments:Initialization
    ✓ should transfer 50M uFragments to the deployer
    ✓ should set the totalSupply to 50M
    ✓ should set the owner
    ✓ should set detailed ERC20 parameters (71ms)
    ✓ should have 9 decimals
    ✓ should have AMPL symbol (43ms)

  Contract: UFragments:setMonetaryPolicy
    ✓ should set reference to policy contract (69ms)
    ✓ should emit policy updated event (46ms)

  Contract: UFragments:setMonetaryPolicy:accessControl
    ✓ should be callable by owner

  Contract: UFragments:setMonetaryPolicy:accessControl
    ✓ should NOT be callable by non-owner
```

```
  Contract: UFragments:PauseRebase
    ✓ should emit pause event
    ✓ should not allow calling rebase
    ✓ should allow calling transfer (59ms)
    ✓ should allow calling approve (59ms)
    ✓ should allow calling allowance
    ✓ should allow calling transferFrom (52ms)
    ✓ should allow calling increaseAllowance (39ms)
    ✓ should allow calling decreaseAllowance (46ms)
    ✓ should allow calling balanceOf
    ✓ should allow calling totalSupply

  Contract: UFragments:PauseRebase:accessControl
    ✓ should be callable by owner (43ms)
    ✓ should NOT be callable by non-owner

  Contract: UFragments:PauseToken
    ✓ should emit pause event
    ✓ should allow calling rebase (50ms)
    ✓ should not allow calling transfer
    ✓ should not allow calling approve
    ✓ should allow calling allowance
    ✓ should not allow calling transferFrom
    ✓ should not allow calling increaseAllowance
    ✓ should not allow calling decreaseAllowance (47ms)
    ✓ should allow calling balanceOf
    ✓ should allow calling totalSupply

  Contract: UFragments:PauseToken:accessControl
    ✓ should be callable by owner (55ms)
    ✓ should NOT be callable by non-owner

  Contract: UFragments:Rebase:accessControl
    ✓ should be callable by monetary policy (42ms)
    ✓ should not be callable by others

  Contract: UFragments:Rebase:Expansion
    ✓ should increase the totalSupply
    ✓ should increase individual balances
    ✓ should emit Rebase
    ✓ should return the new supply (135ms)

  Contract: UFragments:Rebase:Expansion
    when totalSupply is less than MAX_SUPPLY and expands beyond
      ✓ should increase the totalSupply to MAX_SUPPLY
      ✓ should emit Rebase
    when totalSupply is MAX_SUPPLY and expands
      ✓ should NOT change the totalSupply
      ✓ should emit Rebase

  Contract: UFragments:Rebase:NoChange
    ✓ should NOT CHANGE the totalSupply
    ✓ should NOT CHANGE individual balances
    ✓ should emit Rebase

  Contract: UFragments:Rebase:Contraction
    ✓ should decrease the totalSupply
    ✓ should decrease individual balances
    ✓ should emit Rebase

  Contract: UFragments:Transfer
    deployer transfers 12 to A
      ✓ should have correct balances (72ms)
    deployer transfers 15 to B
      ✓ should have balances [973,15] (96ms)
    deployer transfers the rest to C
      ✓ should have balances [0,973] (77ms)
    when the recipient address is the contract address
      ✓ reverts on transfer
      ✓ reverts on transferFrom
    when the recipient is the zero address
      ✓ emits an approval event
      ✓ transferFrom should fail (39ms)

  Contract: UFragmentsPolicy
    ✓ should reject any ether sent to it

  Contract: UFragmentsPolicy:initialize
    initial values set correctly
      ✓ deviationThreshold
      ✓ rebaseLag
      ✓ minRebaseTimeIntervalSec
      ✓ epoch
      ✓ rebaseWindowOffsetSec
      ✓ rebaseWindowLengthSec
      ✓ should set owner
      ✓ should set reference to uFragments

  Contract: UFragmentsPolicy:setMarketOracle
    ✓ should set marketOracle

  Contract: UFragments:setMarketOracle:accessControl
    ✓ should be callable by owner
    ✓ should NOT be callable by non-owner

  Contract: UFragmentsPolicy:setCpiOracle
    ✓ should set cpiOracle (38ms)

  Contract: UFragments:setCpiOracle:accessControl
    ✓ should be callable by owner
    ✓ should NOT be callable by non-owner

  Contract: UFragmentsPolicy:setDeviationThreshold
    ✓ should set deviationThreshold

  Contract: UFragments:setDeviationThreshold:accessControl
    ✓ should be callable by owner
    ✓ should NOT be callable by non-owner

  Contract: UFragmentsPolicy:setRebaseLag
    when rebaseLag is more than 0
      ✓ should setRebaseLag (40ms)
    when rebaseLag is 0
      ✓ should fail

  Contract: UFragments:setRebaseLag:accessControl
    ✓ should be callable by owner
    ✓ should NOT be callable by non-owner

  Contract: UFragmentsPolicy:setRebaseTimingParameters
    when interval=0
      ✓ should fail
    when offset > interval
      ✓ should fail
    when params are valid
      ✓ should setRebaseTimingParameters (68ms)

  Contract: UFragments:setRebaseTimingParameters:accessControl
    ✓ should be callable by owner
    ✓ should NOT be callable by non-owner

  Contract: UFragmentsPolicy:Rebase
    when minRebaseTimeIntervalSec has NOT passed since the previous rebase
      ✓ should fail

  Contract: UFragmentsPolicy:Rebase
    when rate is within deviationThreshold
      ✓ should return 0 (771ms)

  Contract: UFragmentsPolicy:Rebase
    when rate is more than MAX_RATE
      ✓ should return same supply delta as delta for MAX_RATE (737ms)

  Contract: UFragmentsPolicy:Rebase
    when uFragments grows beyond MAX_SUPPLY
      ✓ should apply SupplyAdjustment {MAX_SUPPLY - totalSupply} (87ms)

  Contract: UFragmentsPolicy:Rebase
    when uFragments supply equals MAX_SUPPLY and rebase attempts to grow
      ✓ should not grow (69ms)

  Contract: UFragmentsPolicy:Rebase
    when the market oracle returns invalid data
      ✓ should fail (170ms)
    when the market oracle returns valid data
      ✓ should NOT fail (203ms)

  Contract: UFragmentsPolicy:Rebase
    when the cpi oracle returns invalid data
      ✓ should fail (170ms)
    when the cpi oracle returns valid data
      ✓ should NOT fail (196ms)

  Contract: UFragmentsPolicy:Rebase
    positive rate and no change CPI
      ✓ should increment epoch
      ✓ should update lastRebaseTimestamp
      ✓ should emit Rebase with positive requestedSupplyAdjustment
      ✓ should call getData from the market oracle
      ✓ should call getData from the cpi oracle
      ✓ should call uFrag Rebase

  Contract: UFragmentsPolicy:Rebase
    negative rate
      ✓ should emit Rebase with negative requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    when cpi increases
      ✓ should emit Rebase with negative requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    when cpi decreases
      ✓ should emit Rebase with positive requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    rate=TARGET_RATE
      ✓ should emit Rebase with 0 requestedSupplyAdjustment

  Contract: UFragmentsPolicy:Rebase
    when its 5s after the rebase window closes
      ✓ should fail (149ms)
    when its 5s before the rebase window opens
      ✓ should fail (148ms)
    when its 5s after the rebase window opens
      ✓ should NOT fail (208ms)
    when its 5s before the rebase window closes
      ✓ should NOT fail (214ms)

  Contract: UInt256Lib
    toInt256Safe
      when then number is more than MAX_INT256
        ✓ should fail
      when then number is MAX_INT256
        ✓ converts int to uint256 safely
      when then number is less than MAX_INT256
        ✓ converts int to uint256 safely
      when then number is 0
        ✓ converts int to uint256 safely

  Contract: UFragments:ERC20
    totalSupply
      ✓ returns the total amount of tokens
    balanceOf
      when the requested account has no tokens
        ✓ returns zero
      when the requested account has some tokens
        ✓ returns the total amount of tokens

  Contract: UFragments:ERC20:transfer
    when the sender does NOT have enough balance
      ✓ reverts (43ms)
    when the sender has enough balance
      ✓ should transfer the requested amount (57ms)
      ✓ should emit a transfer event
    when the recipient is the zero address
      ✓ should fail

  Contract: UFragments:ERC20:transferFrom
    when the spender does NOT have enough approved balance
      when the owner does NOT have enough balance
        ✓ reverts (49ms)
      when the owner has enough balance
        ✓ reverts (49ms)
    when the spender has enough approved balance
      when the owner does NOT have enough balance
        ✓ should fail (57ms)
      when the owner has enough balance
        ✓ transfers the requested amount
        ✓ decreases the spender allowance
        ✓ emits a transfer event

  Contract: UFragments:ERC20:approve
    when the spender is NOT the zero address
      when the sender has enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ approves the requested amount and replaces the previous one
          ✓ emits an approval event
      when the sender does not have enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ approves the requested amount
          ✓ emits an approval event

  Contract: UFragments:ERC20:increaseAllowance
    when the spender is NOT the zero address
      when the sender has enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ increases the spender allowance adding the requested amount
          ✓ emits an approval event
      when the sender does not have enough balance
        when there was no approved amount before
          ✓ approves the requested amount
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ increases the spender allowance adding the requested amount
          ✓ emits an approval event

  Contract: UFragments:ERC20:decreaseAllowance
    when the spender is NOT the zero address
      when the sender does NOT have enough balance
        when there was no approved amount before
          ✓ keeps the allowance to zero
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ decreases the spender allowance subtracting the requested amount
          ✓ emits an approval event
      when the sender has enough balance
        when there was no approved amount before
          ✓ keeps the allowance to zero
          ✓ emits an approval event
        when the spender had an approved amount
          ✓ decreases the spender allowance subtracting the requested amount
          ✓ emits an approval event


  168 passing (30s)

-----------------------|----------|----------|----------|----------|----------------|
File                   |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------------|----------|----------|----------|----------|----------------|
 contracts/            |      100 |     97.5 |      100 |      100 |                |
    UFragments.sol       |      100 |      100 |      100 |      100 |                |
    UFragmentsPolicy.sol |      100 |    95.45 |      100 |      100 |                |
 contracts/lib/        |      100 |      100 |      100 |      100 |                |
    SafeMathInt.sol      |      100 |      100 |      100 |      100 |                |
    UInt256Lib.sol       |      100 |      100 |      100 |      100 |                |
-----------------------|----------|----------|----------|----------|----------------|
All files              |      100 |    98.15 |      100 |      100 |                |
-----------------------|----------|----------|----------|----------|----------------|




  Contract: MedianOracle:GasTests
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78447
Initial pushReport() gas: 78319
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63457
Update pushReport() gas: 63329
    when the sources are live
getData() gas: 223817
      ✓ should calculate the combined market rate and volume (313ms)

  Contract: MedianOracle:constructor
    ✓ should fail if a parameter is invalid (264ms)

  Contract: MedianOracle:providersSize
    ✓ should return the number of sources added to the whitelist (182ms)

  Contract: MedianOracle:addProvider
    when successful
      ✓ should emit SourceAdded message
      ✓ should add source to the whitelist
      ✓ should not add an existing source to the whitelist

  Contract: MedianOracle:pushReport
    ✓ should only push from authorized source (156ms)
    ✓ should emit ProviderReportPushed message (143ms)
```

```
Contract: MedianOracle:addProvider:accessControl
   ✓ should be callable by owner (48ms)
   ✓ should NOT be callable by non-owner

Contract: MedianOracle:removeProvider
  when source is part of the whitelist
     ✓ should emit SourceRemoved message
     ✓ should remove source from the whitelist (110ms)

Contract: MedianOracle:removeProvider
   ✓ Remove last element (138ms)
   ✓ Remove middle element (129ms)
   ✓ Remove only element (257ms)

Contract: MedianOracle:removeProvider
   ✓ when source is NOT part of the whitelist (287ms)

Contract: MedianOracle:removeProvider:accessControl
   ✓ should be callable by owner (49ms)
   ✓ should NOT be callable by non-owner

Contract: MedianOracle:getData
  when the reports are valid
     ✓ should calculate the combined market rate and volume (128ms)

Contract: MedianOracle:getData
  when one of reports has expired
     ✓ should emit ReportTimestampOutOfRange message (129ms)
     ✓ should calculate the exchange rate (138ms)

Contract: MedianOracle:getData
  when one of the reports is too recent
     ✓ should emit ReportTimestampOutOfRange message (118ms)
     ✓ should calculate the exchange rate (112ms)

Contract: MedianOracle:getData
  when not enough providers are valid
     ✓ should emit ReportTimestampOutOfRange message (97ms)
     ✓ should not have a valid result (119ms)

Contract: MedianOracle:getData
  when all reports have expired
     ✓ should emit 2 ReportTimestampOutOfRange messages (71ms)
     ✓ should return false and 0 (62ms)

Contract: MedianOracle:getData
  when recent is too recent and past is too old
     ✓ should emit ReportTimestampOutOfRange message (66ms)
     ✓ should fail (51ms)

Contract: MedianOracle:getData
  when recent is too recent and past is too recent
     ✓ should emit ReportTimestampOutOfRange message (80ms)
     ✓ should fail (63ms)

Contract: MedianOracle:getData
  when recent is too recent and past is valid
     ✓ should succeeded (54ms)

Contract: MedianOracle:getData
  when recent is not too recent nor too old
     ✓ should succeed (54ms)

Contract: MedianOracle:getData
  when recent is not too recent but too old
     ✓ should fail (46ms)

Contract: MedianOracle:PurgeReports
   ✓ data not available after purge (51ms)
   ✓ data available after another report (108ms)
   ✓ cannot purge a non-whitelisted provider (134ms)

Contract: Select
  Select:computeMedian
     ✓ median of 1 (46ms)
     ✓ median of 2 (47ms)
     ✓ median of 3 (54ms)
     ✓ median of odd sized list (364ms)
     ✓ median of even sized list (495ms)
     ✓ not enough elements in array
     ✓ median of empty list
     ✓ median of list of size 0


45 passing (16s)

-------------------|----------|----------|----------|----------|----------------|
File               |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-------------------|----------|----------|----------|----------|----------------|
 contracts/        |      100 |      100 |      100 |      100 |                |
   MedianOracle.sol |      100 |      100 |      100 |      100 |                |
 contracts/lib/    |      100 |      100 |      100 |      100 |                |
   Select.sol       |      100 |      100 |      100 |      100 |                |
 contracts/mocks/  |      100 |      100 |      100 |      100 |                |
   SelectMock.sol   |      100 |      100 |      100 |      100 |                |
-------------------|----------|----------|----------|----------|----------------|
All files          |      100 |      100 |      100 |      100 |                |
-------------------|----------|----------|----------|----------|----------------|
```

Automated Analyses

Oyente

Oyente reported integer overflows and underflows in `UFragmentPolicy.sol`, `UFragments.sol`, and `MedianOracle.sol`. We classified all of the reported issues as false positives.

Mythril

Mythril reported external call to fixed address by `UFragmentPolicy.rebase()`. We classified it as a false positive since the call refers to `rebase()` from `UFragments`.

MAIAN

MAIAN reported no issues.

Securify

Securify reported unrestricted writes in `UFragmentPolicy.sol`, `UFragments.sol`, and `MedianOracle.sol`. We classified all of the reported issues as false positives.

Adherence to Specification

The code mostly conforms to the specification; although CPI oracle is present in the code, it is not mentioned in the whitepaper.

Code Documentation

The code is well-documented. The line 59 of the file `MedianOracle.sol`, however, appears to be a copy of the previous line.

Adherence to Best Practices

The code conforms to best practices.

Ampleforth team asked Quantstamp to check whether there are any issues with contracts upgradability. We have found no issues, however:

- it is important that the layout of variables in contracts must remain stable while using `zos` for upgrades.
- we noted that unlike `UFragments.sol` and `UFragmentsPolicy.sol`, `MedianOracle.sol` is not an upgradable contract.

Appendix

File Signatures

The following are the SHA-256 hashes of the audited contracts and/or test files. A smart contract or file with a different SHA-256 hash has been modified, intentionally or otherwise, after the audit. 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 audit.

Contracts

```
81231683bd400eabba5c8a81f98fd71729329833f35b4e6763b37e593ca7f437
./UFragmentsPolicy.sol

6e6d0f4ab07de59b522389faec3645492a1d825e253d0095b2157b1185a1fc70
./UFragments.sol

ea6b6d6a803759c2c786147748ae14a0128548795d7b5b29fe268bc8570526c9
./SafeMathInt.sol

5c6e056051acd09cd87db055fda46b833f23debad96eb1bcf50c889f151b2493
./UInt256Lib.sol

88c3816d362b628f639e9fa11c239c213631304202ab083c7e7bd5c774950d80
./SafeMathIntMock.sol

ce3b4a9189f98f00413a827eee660e551d1f86ebd12fc5f85fcf408eda4603a3
./Mock.sol

311e9b5fc63fc6545c84eb625ebfd28be984ab69e6b40512496b292a4b9d978e
./UInt256LibMock.sol

d55b43570b1101b83fd2a2a7f3ffa6e8649c1bd508fd76460a1aa6d18a4ebdac
./MockUFragments.sol

50778234907d767f712499e0216fdd9c4cc8bc29ac34d9505a4a4084cf9e89b4
./MockOracle.sol

d3aaa982436039f677957a4c08b66dd8e2fd7ce5b0be5de65f3ee63130ef4013
./MedianOracle.sol

6f49d45d993a7949a4d79e144252a373a4db9f7d6f1797cf60e72c026f7dbd69
./Select.sol

eda4a0e2022b2c761a89d204bf49a52c169b43faa5d78c090e928b44e64e0faa
./SelectMock.sol
```

Tests

```
00f345504fc03f57a0efa911f4653c7d63e4b83b4f718530e5c2f6001a852789
./UFragmentsPolicy.js

ea53ce1799f0cc86eb0eebf324ce51eeb025cbbc76fc1ace47d44b1bba7bc94f
./transfer_precision.js

13efe427165805ae6ff8bf6ba55a948774ad25c03e7551e04440b5727c12bb83
./supply_precision.js

ae189446117d98e8c54b4401861d4d367102c37569588782ec7b9dbc22690e2f
./UFragments.js

269c3d332f1fa8169e1fa3fc5edf11705a4a39d264546c3aef0c51d0b5c25867
./UInt256Lib.js

9df063e87c560eb4a82cbfe0b8358b69f0223985afeca8326351ce013380313f
./SafeMathInt.js

545da6a1916c4836ed8c955ae31c908352787dae233cdf9ce56fcbdb32098131
./uFragments_erc20_behavior.js

94d69d5c05a3f052bf4a248a11e2de852041b646f8a2b78bb02472588660d899
./gas_cost.js

191439e9d3b5c601cc77e868d0f6c7cc08424f0242be4716fdcc699ba427c49d
./select.js

0005c1de9bf0ed0dc9acbe11f46d02e24ec1c1526cd9e8db0edcfaba80818acc
./median_oracle.js
```

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.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The Solidity language itself and other smart contract languages remain under development and are subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity or the smart contract programming language, or other programming aspects that could present security risks. You may risk loss of tokens, Ether, and/or other loss. A report is not an endorsement (or other opinion) of any particular project or team, and the report does not guarantee the security of any particular project. A report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. To the fullest extent permitted by law, we disclaim all warranties, express or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked website, or any website or mobile application featured in any banner or other advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. You may risk loss of QSP tokens or other loss. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.