Quantstamp Security Assessment Certificate

May 24th 2021 — Quantstamp Verified

Ichi

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

Executive Summary

Type	Stable Coin and Investment Platform
Auditors	Ed Zulkoski, Senior Security Engineer Christoph Michel, Research Engineer Mohsen Ahmadvand, Senior Research Engineer
Timeline	2021-04-22 through 2021-05-24
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	https://docs.ichi.farm/
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
ichi-oneToken	09fca74 (initial report)
ichi-oneToken	11c210e (revised report)

Total Issues	**29**	(21 Resolved)
High Risk Issues	**4**	(4 Resolved)
Medium Risk Issues	**4**	(3 Resolved)
Low Risk Issues	**11**	(9 Resolved)
Informational Risk Issues	**4**	(2 Resolved)
Undetermined Risk Issues	**6**	(3 Resolved)

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

Summary of Findings

During the audit, 29 issues were identified ranging from High to Undetermined severity. We recommend addressing all issues before using the code in production. The documentation provided was of high quality which facilitated the audit. Although the tests are of high coverage, we recommend adding more tests to cover the issues presented in the findings.
Update: Our report has been updated based on commit 11c210e. All issues have been either resolved or acknowledged.

ID	Description	Severity	Status
QSP-1	Minting fee can be avoided if insufficient member tokens are provided	High	Fixed
QSP-2	Wrong precision returned in `Oracle.amountRequired`	High	Fixed
QSP-3	OneTokens can be redeemed for profit because of decimal issues	High	Fixed
QSP-4	OneTokens can be redeemed 1-to-1 for collateral	High	Fixed
QSP-5	Privileged Roles and Ownership	Medium	Acknowledged
QSP-6	Incorrect `amountRequired` function	Medium	Fixed
QSP-7	`removeStrategy` does not close the strategy's position	Medium	Fixed
QSP-8	Missing pair initializer check in `UniswapOracleSimple`	Medium	Fixed
QSP-9	Stub volatility values	Low	Acknowledged
QSP-10	Setter functions for min and max ratio may break step size invariant	Low	Fixed
QSP-11	`updateMintingRatio` can be called repeatedly	Low	Fixed
QSP-12	Unclear distinction between `execute` and `executeTransaction`	Low	Fixed
QSP-13	`setStrategy` ignores `allowance` parameter	Low	Fixed
QSP-14	Unchecked function arguments	Low	Fixed
QSP-15	`mintingRatio` might never reach its bounds	Low	Fixed
QSP-16	`Incremental`'s events miss oneToken parameter	Low	Fixed
QSP-17	Approval of old strategy is not revoked	Low	Fixed
QSP-18	`setStrategyAllowance` misses funds that are currently deployed	Low	Acknowledged
QSP-19	Incompatible with non-standard ERC20 tokens	Low	Fixed
QSP-20	Clone-and-Own	Informational	Acknowledged
QSP-21	Unlocked Pragma	Informational	Acknowledged
QSP-22	Inconsistent constraints in `setMinRatio` and `setMaxRatio`	Informational	Fixed
QSP-23	Oracle `update` function not always called after initialization	Informational	Fixed
QSP-24	Unclear volatility computation in composite oracles	Undetermined	Acknowledged
QSP-25	`maxOrderVolume` is always `INFINITE`	Undetermined	Acknowledged
QSP-26	Unclear why `init` may be called multiple times	Undetermined	Acknowledged
QSP-27	Contract size potentially exceeds the limit	Undetermined	Fixed
QSP-28	Minting fees are only paid on collateral value	Undetermined	Fixed
QSP-29	Minting does potentially not update the member token oracle	Undetermined	Fixed

Quantstamp Audit Breakdown

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

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

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

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Toolset

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

Setup

Tool Setup:

- Slither v0.7.1
- Mythril v0.22.19

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`

Findings

QSP-1 Minting fee can be avoided if insufficient member tokens are provided

Severity: *High Risk*

Status: Fixed

File(s) affected: `OneTokenV1.sol`

Description: The function `mint` adds a fee to the collateral USD value on L116: `collateralUSDValue = collateralUSDValue.add(collateralUSDValue.mul(mintingFee).div(PRECISION));`. If the member token allowance is too low, the `collateralUSDValue` is recomputed in L125-130 based on the amount of member tokens that the user can provide. However, the fee is never re-applied to this new amount.
Rational minters would always skip paying the fee by providing an allowance of 1 wei less than the required member token value.

Recommendation: Apply the minting fee after the `if-block` on L125-130.

QSP-2 Wrong precision returned in `Oracle.amountRequired`

Severity: *High Risk*

Status: Fixed

File(s) affected: `oracle/OracleCommon.sol`, `oracle/composite/ICHICompositeOracle.sol`, `oracle/pegged/ICHIPeggedOracle.sol`, `oracle/uniswap/UniswapOracleSimple.sol`

Description: It's not clear what precision the `amountUsd` argument of the `amountRequired` function is supposed to be. For `ICHIPeggedOracle`, it always returns the same precision as `amountUsd` which leads to a bug when used in this `OneTokenV1.mint` call:

```
(uint collateralTokensReq, /* volatility */) = IOracle(assets[collateralToken].oracle).amountRequired(collateralToken, collateralUSDValue);
```

`collateralUSDValue` is in `oneToken` precision (18 decimals) but when used with another stablecoin like USDT or USDC (6 decimals) the pegged oracle returns 10^12 times the stable coin amount actually required.
The same issue exists for `UniswapOracleSimple.amountRequired` where it seems like `amountUsd` should be in `indexToken` precision to return the correct amount in `token`.

Recommendation:

- Document the precision of `amountUsd` and fix issues on all call sites. For instance, it could always be in 10^18 precision and the resulting amount needs to be converted to `token` precision. This would not require changes in the `OneTokenV1.mint` oracle call.
- Add mainnet fork tests with tokens of different decimals and the actual tokens that the protocol will end up using.

QSP-3 OneTokens can be redeemed for profit because of decimal issues

Severity: *High Risk*

Status: Fixed

File(s) affected: `OneTokenV1.sol`

Description: The `redeem` function takes an `amount` parameter reflecting oneTokens (18 decimals) to be redeemed. This amount is added to the user's **collateral** balance but the collateral token could have a different number of decimals, like USDT/USDC (6 decimals), netting a 10^12 profit and allowing to withdraw the whole reserve.

```
uint netTokens = amount.sub(amount.mul(redemptionFee).div(PRECISION)); // div by precision because of fee
// increases collateral with oneToken amount
increaseUserBalance(msg.sender, collateral, netTokens);
```

Recommendation:

- Convert the amount to `collateral` decimals and add tests cases
- Add mainnet fork tests with tokens of different decimals and the actual tokens that the protocol will end up using.

QSP-4 OneTokens can be redeemed 1-to-1 for collateral

Severity: *High Risk*

Status: Fixed

File(s) affected: `OneTokenV1.sol`

Description: The `redeem` function updates the `collateral` oracle price but does not consult it for a price. Instead, collateral is redeemed 1-to-1 but collateral could be off-peg.
As `mint` consults the collateral oracle price, but `redeem` treats them as pegged, there is arbitrage potential and the collateral reserve could be emptied.

Recommendation: Consult the collateral oracle to determine the collateral to receive for the oneToken amount. Similar to how `mint` computes `collateralTokensReq`.

QSP-5 Privileged Roles and Ownership

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `Arbitrary.sol`, `OneTokenV1.sol`

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

1. Owners of `Arbitrary` strategy contracts can transfer any tokens anywhere. It is also expected that they will retrieve underlying positions manually before `_closeAllPositions` is invoked.
2. When a user mints oneTokens, they can provide both collateral and member tokens, however when redeeming, only collateral tokens may be obtained. It must be ensured by the contract owner and underlying strategy that assets are used in a principle-preserving way. If the owner/strategy does not preserve the principle, the initial holders will indeed receive a 1-to-1 value exchange until the collateral reserve is depleted. Holders that redeem last will find themselves with *valueless oneTokens* that cannot be redeemed anymore - not even at the minting ratio of 90%.
3. Owners of `OneTokenV1` can set redemption and minting fees up to 100%.

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. Limit the owner's actions to a set of pre-defined transactions where possible.

Update: From the Ichi team: oneTokens are designed to be governed by the members of each oneToken community. As such, the contract can enable strategies for management of the community treasury which can include control by a specified owner. This behavior is included in public facing materials and developer documentation.

QSP-6 Incorrect `amountRequired` function

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `ICHICompositeOracle.sol`

Description: The function appears to be copied from `ICHIPeggedOracle.sol`, however a 1:1 ratio between input and output tokens is not guaranteed in a composite oracle.

Recommendation: Implement `amountsRequired`, which may involve iterating through the interim tokens in reverse.

QSP-7 `removeStrategy` does not close the strategy's position

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `OneTokenV1Base.sol`

Description: There is no call to `closeStrategy(token)` in `removeStrategy`, which may cause tokens to be locked.

Recommendation: The function should invoke `closeStrategy(token)` as is done in `setStrategy`.

Update: From the Ichi team: A closed strategy remains under the control of strategy governance which can close the strategy and transfer assets to the vault. Depending on the number of assets and steps involved, it may not be feasible to close all positions. Positions can be closed individually by token. Further, it may not be feasible to close a given position, e.g. timelock. We have separated the concerns of closing the strategy and recovering the funds in the strategy (See SOL-2) to maximize the probability of eventually recovering all funds in a wide variety of scenarios. Strategy allowance (from vault) is reset to 0 on OneTokenV1Base.closeStrategy. This action is not intended to recover funds.

QSP-8 Missing pair initializer check in `UniswapOracleSimple`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `UniswapOracleSimple.sol`

Description: `UniswapOracleSimple.update` does not check if the pair has been initialized and will store wrong prices as the average prices will be computed since 1970. (time elapsed equals block timestamp.)
The `pairInfo` function will return this wrong price, reading the price through `read`/`consult` will fail though.

Recommendation: Check that `p.token0 != NULL_ADDRESS`.

QSP-9 Stub volatility values

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `UniswapOracleSimple.sol`

Description: The volatility of any Uniswap pair is reported as zero, even though price fluctuations may exist.

Recommendation: Incorporate volatility logic or document why it is set to zero for Uniswap pairs.

Update: From the Ichi team: The volatility parameter isn't used at this time but it is included in the Oracle response ABI to facilitate the future construction of Oracles that detect price volatility. This is to facilitate future use-cases. For example, a future MintMaster might use the volatility metric to consider adjusting the minting ratio or setting a maximum order volume. OneTokenV1 respects both indicators.

QSP-10 Setter functions for min and max ratio may break step size invariant

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Incremental.sol`

Description: The function `setStepSize` ensures that it "must be smaller than the difference of min and max". However, `setMinRatio` and `setMaxRatio` do not have similar checks. If the difference is set too small, this invariant will no longer hold.

Recommendation: Add checks to `setMinRatio` and `setMaxRatio`. Either revert if the new parameters would make the step size too large, or adjust the step size appropriately.

QSP-11 `updateMintingRatio` can be called repeatedly

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Incremental.sol`

Description: Although the function is only intended to make a small step to the ratio value, the function can be called repeatedly (even in the same transaction). This could potentially be used if an arbitrage attack could benefit from changing the ratio significantly beyond the step size.

Recommendation: Consider limiting the access control of `updateMintingRatio`. Another option is to only allow a step update once per block (i.e., the function would simply `return` on subsequent calls in the same block).

QSP-12 Unclear distinction between `execute` and `executeTransaction`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `StrategyCommon.sol`, `Arbitrary.sol`, `OneTokenV1Base.sol`

Description: `StrategyCommon` declares a virtual function `execute`, whereas `Arbitrary` declares `executeTransaction`. It is further unclear how `Arbitrary.executeTransaction` relates to `OneTokenV1Base.executeStrategy`, which invokes `IStrategy(strategy).execute();`. If the strategy is of type `Arbitrary`, it appears that this function call will do nothing.
Note that `OneTokenV1Base.executeStrategy` also emits a `StrategyExecuted` event, so there may be duplicate events emitted. While the `Arbitrary.executeTransaction` function does not emit a `StrategyExecuted` event, the inherited `execute` function does.

Recommendation: Clarify the strategy execution logic, and how it relates to `OneTokenV1Base`.

Update: From the Ichi team: The distinction can be summarized as "Execute the Strategy" which all Strategies must implement even if there is no automation, and "Execute this Transaction" which Arbitrary offers in addition to the mandatory function.
Every Strategy must have an execute() function that the assigned Controller calls periodically (triggered by oneToken redemption). execute() runs the automation that the Strategy contract implements. The design is silent on the details of automation implemented by a Strategy, but it could include automated investment, rebalancing, transferring funds to other strategies and so on. The default execute() function in StrategyCommon does nothing. Event emission has been removed to help clarify that nothing was done in the case that the default execute function isn't overwritten by a Strategy implementation.
The Arbitrary strategy is a special case. It's executeTransaction() function is the function governance calls to execute an arbitrary transaction. Arbitrary does indeed do nothing when the Controller calls its mandatory execute() function because no automation is intended. It's governance-only function, executeTransaction(), is adjacent to the minimal interface that all Strategies must implement. The expected development of many modular Strategies will likely negate the need to use Arbitrary Strategy, over time. In the meantime, it provides a flexible solution that is compatible with the new modular structure.
Two levels of governance approval are required to adopt an "Arbitrary" Strategy. It must be accepted at the Factory level (ICHI Governance) and then assigned to an asset in a vault by OneToken Governance.

QSP-13 `setStrategy` ignores `allowance` parameter

Severity: *Low Risk*

Status: Fixed

File(s) affected: `OneTokenV1Base.sol`

Description: The function always sets the allowance to `INFINITE`: `IStrategy(strategy).setAllowance(token, INFINITE);`, regardless of the `allowance` parameter.

Recommendation: Set the allowance based on the function parameter.

QSP-14 Unchecked function arguments

Severity: *Low Risk*

Status: Fixed

File(s) affected: `OracleCommon.sol`, `UniswapOracleSimple.sol`, `ICHICompositeOracle.sol`, `ICHIModuleCommon.sol`, `Incremental.sol`, `OneTokenV1Base.sol`, `OneTokenFactory.sol`

Description: The following functions should perform additional checks on their arguments:

1. `OracleCommon.constructor` should check that `oneTokenFactory_` and `indexToken_` are non-zero.
2. In `UniswapOracleSimple.constructor`, `indexToken_` and `period_` should be non-zero. This would also handle the same missing check in `ICHICompositeOracle.constructor`.
3. `ICHIModuleCommon.updateDescription` does not check if `description` length is non-zero, but the constructor does.
4. `Incremental._updateMintingRatio` should check `p.set`.
5. `OneTokenV1Base.setStrategyAllowance` should check if `assets[token].strategy` exists. Otherwise, a null strategy is approved.
6. `OneTokenFactory.assignOracle` does not check `isModule(oracle)` but `admitForeignToken` does.

Recommendation: Add the corresponding `require` statements to each function.

QSP-15 `mintingRatio` might never reach its bounds

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Incremental.sol`

Description: The `mintingRatio` might never reach its min or max ratio as it is only changed by `stepSize` and not clamped, see `Incremental.getMintingRatio4`.

Recommendation: Change the ratio by `stepSize` and clamp it to min/max ratio if it is out of bounds. Choose a stepSize that divides `max-min`.

QSP-16 `Incremental`'s events miss oneToken parameter

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Incremental.sol`

Description: `Incremental` allows changing settings for several oneTokens. However, its events don't indicate the `oneToken` the settings were changed for which makes it hard for a backend to associate settings changes with the `oneToken`.

Recommendation: Add the `oneToken` to the events.

QSP-17 Approval of old strategy is not revoked

Severity: *Low Risk*

Status: Fixed

File(s) affected: `OneTokenV1Base.sol`

Description: `OneTokenV1Base.closeStrategy` should revoke the approval of an old strategy (similar to how `setStrategy` sets approval) to shut down vulnerable strategies from causing havoc. The `@dev` comment talks about resetting the approval but it's not done:

> @dev strategy remains assigned the asset with allowance set to 0.

Recommendation: Reset the approval.

QSP-18 `setStrategyAllowance` misses funds that are currently deployed

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `OneTokenV1Base.sol`

Description: The `setStrategyAllowance` functions computes the current holdings as the balance of the strategy contract: `IERC20(token).balanceOf(a.strategy);`. This does not include the holdings that are already used in a different contract as part of the strategy.
This is also susceptible to being front-run with a transaction that moves the funds to a different contract in the strategy but as strategy funds are controlled by governance, this will most likely not be an issue.

Recommendation: Add a function to strategies that returns all funds currently held and deployed.

Update: From the Ichi team: We have changed the flow to support the increase/decrease allowance pattern. Since a full accounting would be complex and expensive, possibly infeasible, we do not adjust the allowance instructions from Governance.

QSP-19 Incompatible with non-standard ERC20 tokens

Severity: *Low Risk*

Status: Fixed

File(s) affected: `OneTokenV1.sol, StrategyCommon.sol`

Description: The documentation mentions using stable coins as collateral but some common ERC20 stablecoins, like USDT, do not correctly implement the ERC20 specification as the functions don't return a boolean. This will lead to reverts when interacting with them, for instance here in `OneTokenV1.mint`:

`IERC20(collateralToken).transferFrom`

Recommendation: Use OpenZeppelin's `SafeERC20` library in all contracts if support for USDT is required. If not, make sure to check the return value of the ERC20 function calls.

QSP-20 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `_openzeppelin/*, _uniswap/*`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

Update: From the Ichi team: We adopted the explicit strategy employed to disambiguate the exact code needed to recreate the deployed bytecode in the future. Changing the structure at this stage could be extensive and error-prone so we have decided to leave the structure in its original form.

QSP-21 Unlocked Pragma

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Several files`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

Update: Pragmas of all contracts to deploy and our own inherited modules are all locked. Open Source pragmas remain unlocked to avoid modification to the files where modification isn't necessary. There is no case where a deployed contract is not bound to 0.7.6.

QSP-22 Inconsistent constraints in `setMinRatio` and `setMaxRatio`

Severity: *Informational*

Status: Fixed

File(s) affected: `Incremental.sol`

Description: The function `setMinRatio` has the constraint `minRatio <= p.maxRatio`, whereas `setMaxRatio` uses `maxRatio > p.minRatio`. It is unclear from these two functions whether `maxRatio == p.minRatio` should be allowed.

Recommendation: Ensure consistent constraints between the two functions.

QSP-23 Oracle `update` function not always called after initialization

Severity: Informational

Status: Fixed

File(s) affected: `OneTokenV1Base.sol`, `OneTokenFactory.sol`

Description: In `OneTokenFactory.sol`, the function `assignOracle` invokes the Oracle's `init` function followed by `update`. However, `admitForeignToken` only invokes `init`.
A similar pattern occurs in `OneTokenV1Base.sol`, where `changeMintMaster` does not invoke `update` unlike in `init`.

Recommendation: Clarify if this is intended or add the `update` call to `admitForeignToken`.

QSP-24 Unclear volatility computation in composite oracles

Severity: Undetermined

Status: Acknowledged

File(s) affected: `ICHICompositeOracle.sol`

Description: The function `read` states that "volatility is calculated by the final oracle". It is not clear why this is only a function of the volatility of the final oracle. If there is volatility in each of the interim tokens, this would influence the resulting `amountOut`.

Recommendation: Consider a volatility function that incorporates all interim volatilities.

Update: From the Ichi team: ICHICompositeOracle calculates volatility as the product of interim volatility detected. Volatility is bound to the range 1-n (absolute value). No Oracle should return volatility: 0.
Response: Note that in future iterations this may need to be normalized. For example, if the range of volatilities is `1,n`, then the multiplication here may exceed `n`.

QSP-25 `maxOrderVolume` is always `INFINITE`

Severity: Undetermined

Status: Acknowledged

File(s) affected: `Incremental.sol`

Description: In the function `getMintingRatio4`, `maxOrderVolume` is set to `INFINITE` in all paths. It is not clear if this is correct or not.

Recommendation: Clarify if the function is implemented correctly. Add inline documentation.

Update: From the Ichi team: The orderVolume metric is for future use-cases, where future MintMaster implementations may clamp down on maximum order size algorithmically, e.g. when volatility is detected. OneTokenV1.sol rejects orders above maxOrderVolume. We have added a governance-controlled manual maxOrderVolume setter in Incremental.sol.

QSP-26 Unclear why `init` may be called multiple times

Severity: Undetermined

Status: Acknowledged

File(s) affected: `StrategyCommon.sol`

Description: The docstring for `init` indicates that "a strategy is dedicated to exactly one oneToken instance and must be re-initializable". It is not clear why it must be re-initializable, given that it is dedicated to a `oneToken` that cannot change.

Recommendation: Clarify why the strategy must be re-initializable.

Update: From the Ichi team: The init flow allows the Strategy to initialize/reset any variables it wants to, consult Oracles or, indeed, any operation required to get its internal affairs in order when it is assigned to an asset in a vault. Initialization may occur more than once from the same vault/asset source. For example, Strategy A assigned, Strategy B assigned, Strategy A re-assigned and re-initialized.

QSP-27 Contract size potentially exceeds the limit

Severity: Undetermined

Status: Fixed

File(s) affected: `OneTokenV1.sol`

Description: The contract `OneTokenV1.sol` appears to exceed 24576 bytes, and may have issues during deployment without further optimization.

Recommendation: Consider enabling the optimizer (with a low "runs" value), turning off revert strings, or using libraries.

Update: From the Ichi team: We compile with 200 optimization cycles and find it deployable. Please advise if this is not the case.

QSP-28 Minting fees are only paid on collateral value

Severity: Undetermined

Status: Fixed

File(s) affected: `OneTokenV1.sol`

Description: The minting fee in `mint` is only paid on the collateral value, not on the member token value, i.e., not on the whole oneToken value.

Recommendation: Clarify the expected behavior.

QSP-29 Minting does potentially not update the member token oracle

Severity: *Undetermined*

Status: Fixed

File(s) affected: `OneTokenV1.sol`

Description: In `mint` there is a comment saying:

```
// this will also update the member token oracle price history
(uint mintingRatio, uint maxOrderVolume) = updateMintingRatio(collateralToken);
```

This is not true unless `MintMaster.oneTokenOracles[oneToken]` always equals the member token oracle defined in the version contract `OneTokenV1.assets[memberToken].oracle`).
The oracle is not updated before performing the `IOracle(assets[memberToken].oracle).amountRequired(memberToken, memberTokensUSDValue)` call which could allow minting undercollateralized oneTokens and redeeming them for a profit.

Recommendation: Update the member oracle or clarify if the contracts will always be initialized such that these two oracle variables will be the same.

Automated Analyses

Slither

In `StrategyCommon.sol` and `OneToken*.sol` several ERC20-related return values are not checked. We recommend using OpenZeppelin's SafeERC20 functions for calls to `transfer`, `transferFrom`, and `approve`.

Mythril

Mythril reports several potential issues with external calls to user-supplied addresses within proxy-related contracts, however we classified these as false positives.

Code Documentation

1. On L18 of `ICHICompositeOracle.sol`, there is a typo "declation". **Update:** fixed.
2. The docstring for `ICHICompositeOracle.constructor` is missing the "@param" description for `interimTokens_`. **Update:** fixed.
3. On L47 of `UniswapOracleSimple.sol`, "registed" should be "registered", and it is not clear what "useToken" refers to (should this be "oneToken"?). **Update:** fixed.
4. On L111 of `UniswapOracleSimple.sol`, "historym" should be removed. **Update:** fixed.
5. On L65 of `OneTokenV1Base.sol`, "mintMast" should be "mintMaster". **Update:** fixed.
6. In `OneTokenV1Base.sol.fromStrategy`, the comment "Relies on allowance" is not correct -- this function also works without the allowance. **Update:** fixed.

Adherence to Best Practices

1. Some string literals are declared in multiple places, e.g., "ICHI V1 Controller" and "ICHI V1 Oracle Implementation". Import declared constants when possible. **Update:** String literals are sometimes separated by design. The Module Types are to catch developer and governance errors on admission.
2. Favor using `uint256` instead of `uint` throughout. **Update:** fixed.
3. In `OracleCommon.sol`, the event `OracleUpdated` is not used anywhere in the project, and could likely be removed. **Update:** fixed.
4. It is unclear why `OneTokenProxyAdmin.sol` is needed when it inherits from `ProxyAdmin` and adds no functionality. **Update:** this is just a wrap up for code organization purposes.
5. Many local variables shadow inherited names of variables or functions and should be renamed: `IOneTokenV1Base.setStrategy.allowance`, `MintMasterCommon.constructor.oneTokenFactory`, `Arbitrary.constructor.oneToken`, `NullStrategy.constructor.oneTokenFactory`, `NullStrategy.constructor.oneToken`, `TestOracle.constructor.oneTokenFactory`, `OneTokenV1Base.setStrategy.allowance`. **Update:** fixed.
6. Many public functions could be declared external, e.g., the `Incremental.set*` functions. **Update:** fixed.
7. `OneTokenV1Base.balances` could check if `strategy == 0` and short circuit to save gas; see `OneTokenV1.getHoldings`. **Update:** fixed.
8. `ControllerCommon` can inherit from `ICHIModuleCommon`. **Update:** ControllerCommon doesn't need to inherit from ICHIModuleCommon. In practice we have found it better to treat them as somewhat unique.

Test Results

Test Suite Results

```
Contract: AddressSet
   ✓ should return the count
   ✓ should not allow to remove unexisting key
   ✓ should remove the first key
   ✓ should remove the last key
   ✓ should remove the middle key
Contract: Arbitrary strategy
   ✓ should be ready to test
   ✓ should be constructed with one token
   ✓ should have 0 allowance before init
```

```
    ✓ should be failed with random token (123ms)
    ✓ should be able to init (272ms)
    ✓ instance cannot be shared between oneTokens (75ms)
    ✓ should not be able to init by a non oneToken user
    ✓ should not be able to call execute by not owner/controller
    ✓ setAllowance to non-zero (78ms)
    ✓ setAllowance to zero (113ms)
    ✓ close all positions (261ms)
    ✓ to vault (55ms)
    ✓ governance could increase and descrease strategy allowance (281ms)
    ✓ from vault (306ms)
    ✓ from vault original (130ms)
    ✓ execute transaction without signature (51ms)
    ✓ execute transaction with signature
    ✓ access control should follow oneToken change of ownership (206ms)
  Contract: Controller
    ✓ should be ready to test
    ✓ should fail to init when initialized from outside
    ✓ should emit Initialized event when initialized by OneToken contract
    ✓ should be able to assing the same controller to multiple oneTokens (67ms)
    ✓ should emit Periodic event (65ms)
    ✓ should call executeStrategy (111ms)
  Contract: Factory
    ✓ should be ready to test
    ✓ should enumerate the factory state (159ms)
    ✓ should update module metadata (101ms)
    ✓ should admit a module (264ms)
    ✓ should remove a module (51ms)
    ✓ should assign and remove an oracle (508ms)
    ✓ should transfer governance (41ms)
    ✓ should guard governance functions (146ms)
    ✓ should deploy a token (1055ms)
    ✓ should update foreign token (39ms)
    ✓ should remove foreign token (45ms)
  Contract: ICHICompositeOracle
    ✓ description is required
    ✓ should be ready to test
    ✓ should be constructed (82ms)
    ✓ can be initialized with oracles in the chain initialize as well (107ms)
    ✓ should be able to update
    ✓ should be configured
    ✓ read should return proper value
    ✓ amountRequired should return proper value
  Contract: ICHICompositeOracleDecimals
    ✓ should be able to update (658ms)
    ✓ should be configured
    ✓ read should return proper value (59ms)
    ✓ amountRequired should return proper value (69ms)
    ✓ should update ratios in interim oracles (305ms)
    ✓ read should return proper value with new LP ratios (65ms)
    ✓ amountRequired should return proper value with new LP ratios (70ms)
  Contract: ICHIPeggedOracle
    ✓ should be ready to test
    ✓ should emit event when being deployed (86ms)
    ✓ initialized from the Factory on foreignToken admission (94ms)
    ✓ should update the module description (39ms)
    ✓ read should return equivalent amount of index tokens for an amount of baseTokens
    ✓ amountRequired should return the tokens needed to reach a target usd value
  Contract: exploratory Integration tests
    ✓ Oracle simple should work for the memberToken (357ms)
  Contract: Integration tests
    ✓ should be deployed
    ✓ Bob should be able to mint one Token with Pegged Oracle for member (98ms)
    ✓ Bob should be able to mint one Token with Uniswap Oracle simple (634ms)
    ✓ Bob transfers some of her oneTokens to Alice (554ms)
    ✓ Oracle price adjustment, member token UP (522ms)
    ✓ Bob redeems tokens, gets USD (512ms)
  Contract: Integration tests with 6/9 decimals
    ✓ should be deployed
    ✓ Bob should be able to mint one Token with Pegged Oracle for member (85ms)
    ✓ Bob should be able to mint one Token with Uniswap Oracle simple (739ms)
    ✓ Bob transfers some of her oneTokens to Alice (645ms)
    ✓ Oracle price adjustment, member token UP (494ms)
    ✓ UniswapOracleSimple read returns correct prices
    ✓ UniswapOracleSimple amountRequired returns correct prices
    ✓ Bob redeems tokens, gets collateral (673ms)
    ✓ UniswapOracleSimple read returns correct prices with 1:2 LP ratio (343ms)
    ✓ UniswapOracleSimple amountRequired returns correct prices with 1:2 LP ratio (321ms)
    ✓ memberToken oracle update update should be called from mint function (523ms)
  Contract: MintMaster
    ✓ should be ready to test
    ✓ should not be able to re-init the mint master from outside of OneToken contract
    ✓ should not be able to update non-initialized Incremental mint master
    ✓ should emit Initialized event when initialized by OneToken contract (156ms)
    ✓ should be able to assing the same mintManster to multiple oneTokens (231ms)
    ✓ should update params (500ms)
    ✓ should adjust the minting ratio (647ms)
    ✓ should update the step size (404ms)
    ✓ should clamp rate at the top and bottom (184ms)
    ✓ access control should follow oneToken change of ownership (117ms)
    ✓ configuration changes one instance should not interfere with the configuration of any other instance (48ms)
  Contract: OneToken V1 Admin
    ✓ should be ready to test
    ✓ should set the redemption fee (47ms)
    ✓ should adjust the minting ratio (191ms)
    ✓ should report holdings of any token (40ms)
  Contract: OneToken V1 Base
    ✓ should be ready to test
    ✓ should know the oneToken factory
    ✓ should know the oneToken's controller
    ✓ should know the oneToken's mintMaster
    ✓ should know the oneToken's member token
    ✓ should already be initialized and reject re-initialization
    ✓ should trasfer governance (97ms)
    ✓ should allow a change of controller (125ms)
    ✓ should allow a change of factory (84ms)
    ✓ should allow a change of mintMaster (426ms)
    ✓ should permit adding an asset (269ms)
    ✓ should permit removing an asset (188ms)
    ✓ should return correct number of collateral tokens (94ms)
    ✓ should permit adding a non-collateral (other) asset/token (227ms)
    ✓ should return correct number of non-collateral (other) tokens (189ms)
    ✓ should allow set, fund, defund and remove a strategy and funds recovery (1367ms)
    ✓ should set a strategy allowance (580ms)
    ✓ should close all strategy positions (472ms)
    ✓ should change a existing strategy (299ms)
    ✓ strategy allowance should be set to 0 when it's removed or re-set (717ms)
    ✓ should not allow setting strategy alowance when strategy is missing
  Contract: OneToken V1 Main
    ✓ should be ready to test
    ✓ should return 0 as initial user balance for tokens
    ✓ should not able to withdraw when balance is 0 (41ms)
    ✓ should be able to mint (739ms)
    ✓ should be able to redeem (232ms)
    ✓ should be able to set minting fee (103ms)
    ✓ should revert on set minting fee if not between 0 and 100 percents (39ms)
    ✓ should apply minting fee correctly in all cases (343ms)
    ✓ should adhere to mintMaster's maxOrder setting (296ms)
  Contract: UniswapOracleSimple
    ✓ should be ready to test
    ✓ should emit event when being deployed
    ✓ should know the oneTokenFactory
    ✓ should know the uniswapV2Factory
    ✓ should know the uniswapOracleSimple
    ✓ should create the pair uniswapV2Factory
    ✓ should fail to init when initialized from outside
    ✓ should fail to init when the pair has no liquidity (145ms)
    ✓ should sync the pair after mint
    ✓ calling update before init should not fail, but it also shouldn't do anything (41ms)
    ✓ always initialized when admitted to factory (96ms)
    ✓ first call for consult should revert (39ms)
    ✓ first call for amountRequired should revert (40ms)
    ✓ consult should return not 0 after update (143ms)
    ✓ read should return not 0
    ✓ amountRequired should return not 0
    ✓ should return pair info (182ms)
    ✓ should fail to create UNI oracle with bad input parameters (104ms)
    ✓ multiple oneToken implementations can share an Oracle (261ms)
    ✓ should be able to switch from Pegged oracle to UniswapOracleSimple (655ms)
    ✓ one oracle can manage multiple currency quotes (148ms)
```

```
    ✓ shared oracle handling N currency pairs (133ms)
    ✓ always updated when selected by oneToken for any foreign token in the vault (383ms)
    ✓ could deploy and admit multiple instances (389ms)
  148 passing (1m)
```

Code Coverage

The code is well covered by the test suite.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.13	94.87	92.86	97.14	
Migrations.sol	0	0	0	0	9,13,17
OneTokenFactory.sol	100	97.37	100	100	
OneTokenProxy.sol	100	100	100	100	
OneTokenProxyAdmin.sol	100	100	100	100	
contracts/_openzeppelin/access/	80	50	80	81.82	
Ownable.sol	80	50	80	81.82	59,60
contracts/_openzeppelin/math/	33.33	20	38.46	40.63	
SafeMath.sol	33.33	20	38.46	40.63	… 192,211,212
contracts/_openzeppelin/proxy/	40.54	21.43	47.83	45.45	
Proxy.sol	100	100	100	80	72
ProxyAdmin.sol	0	0	0	0	… 41,52,63,75
TransparentUpgradeableProxy.sol	37.5	16.67	40	42.11	… 110,121,122
UpgradeableProxy.sol	66.67	50	75	72.73	33,66,67
contracts/_openzeppelin/token/ERC20/	100	50	100	100	
IERC20.sol	100	100	100	100	
SafeERC20.sol	100	50	100	100	
contracts/_openzeppelin/utils/	38.46	18.75	38.46	37.93	
Address.sol	37.5	18.75	36.36	38.46	… 176,180,185
Context.sol	50	100	50	33.33	21,22
contracts/_uniswap/lib/contracts/libraries/	8.02	7.29	23.53	8.39	
Babylonian.sol	0	0	0	0	… 48,49,50,51
BitMath.sol	0	0	0	0	… 78,79,81,83
FixedPoint.sol	25	19.44	36.36	25.49	… 142,143,144
FullMath.sol	0	0	0	0	… 44,46,48,49
contracts/_uniswap/v2-**core/contracts/**	87.18	43.75	78.57	87.5	
UniswapV2Factory.sol	70.59	30	40	72.22	22,43,44,48,49
UniswapV2Pair.sol	100	66.67	100	100	
contracts/_uniswap/v2-**core/contracts/interfaces/**	100	100	100	100	
IUniswapV2Factory.sol	100	100	100	100	
IUniswapV2Pair.sol	100	100	100	100	
contracts/_uniswap/v2-**core/contracts/libraries/**	40	0	40	40	
UQ112x112.sol	100	100	100	100	
UniSafeMath.sol	0	0	0	0	9,13,17
contracts/_uniswap/v2-**periphery/contracts/libraries/**	33.33	10.71	35.71	33.33	
UniswapSafeMath.sol	0	0	0	0	13,17,21
UniswapV2Library.sol	19.44	10	33.33	19.44	… 84,85,86,87
UniswapV2OracleLibrary.sol	**100**	**50**	**100**	**100**	
contracts/common/	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
ICHICommon.sol	100	100	100	100	
ICHIModuleCommon.sol	100	100	100	100	
contracts/controller/	100	100	100	100	
ControllerCommon.sol	100	100	100	100	
contracts/controller/null **controller/**	100	100	100	100	
NullController.sol	100	100	100	100	
contracts/interface/	100	100	100	100	
IController.sol	100	100	100	100	
IERC20Extended.sol	100	100	100	100	
IICHICommon.sol	100	100	100	100	
IICHIOwnable.sol	100	100	100	100	
IMintMaster.sol	100	100	100	100	
IModule.sol	100	100	100	100	
IOneTokenFactory.sol	100	100	100	100	
IOneTokenV1.sol	100	100	100	100	
IOneTokenV1Base.sol	100	100	100	100	
IOracle.sol	100	100	100	100	
IStrategy.sol	100	100	100	100	
InterfaceCommon.sol	100	100	100	100	
contracts/lib/	100	100	100	100	
AddressSet.sol	100	100	100	100	
contracts/mintMaster/	100	50	100	100	
MintMasterCommon.sol	100	50	100	100	
contracts/mintMaster/legacy/	98.7	96.15	100	98.65	
Incremental.sol	98.7	96.15	100	98.65	184
contracts/oracle/	100	80	100	100	
OracleCommon.sol	100	80	100	100	
contracts/oracle/composite/	100	100	100	100	
ICHICompositeOracle.sol	100	100	100	100	
contracts/oracle/pegged/	100	100	100	100	
ICHIPeggedOracle.sol	100	100	100	100	
contracts/oracle/uniswap/	100	87.5	100	100	
UniswapOracleSimple.sol	100	87.5	100	100	
contracts/oz_modified/	80	65.38	73.33	80.88	
ICHIERC20.sol	82.93	50	72.22	82.93	… 189,207,208
ICHIERC20Burnable.sol	0	100	0	0	21,36,38,39
ICHIInitializable.sol	100	87.5	100	100	
ICHIOwnable.sol	81.82	66.67	85.71	83.33	73,74
contracts/strategy/	100	93.75	100	100	
StrategyCommon.sol	100	93.75	100	100	
contracts/strategy/arbitrary/	100	100	100	100	
Arbitrary.sol	100	100	100	100	
contracts/strategy/nullStrategy/	100	100	100	100	
NullStrategy.sol	100	100	100	100	
contracts/testToken/	**56.19**	**22.5**	**60.47**	**56.19**	
CollateralToken.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
DummyMintMaster.sol	8.33	100	9.09	7.69	… 124,135,145
MemberToken.sol	100	100	100	100	
SetTest.sol	100	100	100	100	
TestController.sol	50	100	66.67	50	26
TestMintMaster.sol	50	18.42	64.29	50.82	… 238,239,240
TestOracle.sol	80	100	83.33	75	73,74,75
Token18.sol	100	100	100	100	
Token6.sol	100	100	100	100	
Token9.sol	100	100	100	100	
contracts/version/v1/	100	92.55	100	100	
OneTokenV1.sol	100	100	100	100	
OneTokenV1Base.sol	100	89.71	100	100	
All files	**70.46**	**58.67**	**73.16**	**71.19**	

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

```
05a71ddabd9657181c090128c0842611ddeccb368ce802708289490efb010477 ./contracts/Migrations.sol

fe7300a73eb65dba1d70c83159ff3b62f44afd670952ce7a842eedcb3714e570 ./contracts/OneTokenFactory.sol

a1b714bac21e56c78af96a88dd2b299c772b48ee6a97c252860dbba534f49b3c ./contracts/OneTokenProxy.sol

1d830b9882cd7ced06e52041a79cf74ed6ab8fa6c281351b8b74999eab9de187 ./contracts/OneTokenProxyAdmin.sol

1c0f78ed111bc65fdde96c007562a6ccc82d864540d4a1a768b56d864d07fa88 ./contracts/_uniswap/v2-periphery/contracts/libraries/UniswapSafeMath.sol

91a9c6eb15d6efc4dc244137bda084e3cc300f078bd1c35349fef7f580b3d662 ./contracts/_uniswap/v2-periphery/contracts/libraries/UniswapV2Library.sol

57489035f28c8b057d4420bc2fa697c722f9525c4adede7a0413df7416864392 ./contracts/_uniswap/v2-
periphery/contracts/libraries/UniswapV2OracleLibrary.sol

6ca4b45554f26bfce1d3aa472bb7a765553b9befc77398020c17c587c5cc14c0 ./contracts/_uniswap/v2-core/contracts/UniswapV2Factory.sol

6c48b360711c981dc3b4a31352b2151cd4d138e46a6640029b44caa6eea62906 ./contracts/_uniswap/v2-core/contracts/UniswapV2Pair.sol

7cd1ef83f7c325c4fca1df55f7b635cc9766218515494d99640eba98541d75ab ./contracts/_uniswap/v2-core/contracts/libraries/UniSafeMath.sol

dcda5593ed793f956be9a526b16a20325a13cfc0322bc8602389d0357a1b3645 ./contracts/_uniswap/v2-core/contracts/libraries/UQ112x112.sol

15cb13731b0cafa18b843af7b049a1528e735f25fd8db72bf962ee035c38ede2 ./contracts/_uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol

f26928397a08cfc0d1a81af38e74fa4b5f01993837d59757ec910d3992189642 ./contracts/_uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol

fad5a3f3ff7b5f76bea8fee540d6cbb1e3bf238fef664a83869652a251c8a18e ./contracts/_uniswap/lib/contracts/libraries/Babylonian.sol

738fb85edb0ab8888d5e094f7f769c0a3e71fd546bc828f824f05f819b871367 ./contracts/_uniswap/lib/contracts/libraries/BitMath.sol

1c48ccde5c340c00c13808b065469af43f75d633ebd65f86881d67cffca69544 ./contracts/_uniswap/lib/contracts/libraries/FixedPoint.sol

6527db8c387485b64eb83f3b242970473410bf7a35f4d7db57aec61101fb5bc8 ./contracts/_uniswap/lib/contracts/libraries/FullMath.sol

11ad5e3e21434e00c4ceba1f5a977b7a68bdd7d16b849276ce4ff4495129eec7 ./contracts/_openzeppelin/utils/Address.sol

9a3d1e5be0f0ace13e2d9aa1d0a1c3a6574983983ad5de94fc412f878bf7fe89 ./contracts/_openzeppelin/utils/Context.sol

0573c2961569aa4906845d0cd428b5b7394956170054ceeaa8f8af96cd44875c ./contracts/_openzeppelin/token/ERC20/IERC20.sol

eea796f96939e7dd6119de00941717ce5772d8a942b184f5e48d2b962d0ab655 ./contracts/_openzeppelin/token/ERC20/SafeERC20.sol

c795b86880d9694a8a8dc3fbf4cc6b268d6e35be227a890e8fefed490772d19c ./contracts/_openzeppelin/proxy/Proxy.sol

86e59ef278b5a933a6d056d9a88ddbcb8507ad9d6bb90fa509498294088ae60b ./contracts/_openzeppelin/proxy/ProxyAdmin.sol

07c6ff04d2eaa43df7cdf000769fb1efd7acb05a98ffd37e02c2726e5e0ed204 ./contracts/_openzeppelin/proxy/TransparentUpgradeableProxy.sol

5c4ff5fb61284ed6221b9aa24cfd0bb7eb26da8b3581eddc62796fe0d841baca ./contracts/_openzeppelin/proxy/UpgradeableProxy.sol

4a04d0a20a19e3ef1dcabae9cad9ba006430a4e7eec4d9b519db87999722c98a ./contracts/_openzeppelin/math/SafeMath.sol

3f18ad95b488712604822628a3690af3126cbbe2502caa9d888736777a3b5749 ./contracts/_openzeppelin/access/Ownable.sol

13b7cd295ca96128418785ea8e77cce661d20b94ad227649dcbb0913909a1185 ./contracts/version/v1/OneTokenV1.sol

7df3247ef450431cc03b2cc533b7cbfff2fb8a657ba1184955d978e8078d3413 ./contracts/version/v1/OneTokenV1Base.sol

37ec93512165fd2c5d3dbb6001d6092cddf40a74a13ee2c8fd33a8749bd66af8 ./contracts/testToken/CollateralToken.sol

452977b4ebc0e616b7d321f343ddef1e27061a0c03137aa3c0fb2a7239006616 ./contracts/testToken/DummyMintMaster.sol

1ef5ddefd825bc05ed606d68e388426afc4382b88319108d8e657659a647d482 ./contracts/testToken/MemberToken.sol
```

fa3857aef9761109e82a1523edc444a79f09d29c7a906fa046f03513535cefee ./contracts/testToken/SetTest.sol

1b10ce91ddc302c2742ff5cf8abfa12c4d67e4842226498465a7bc7592fc6b49 ./contracts/testToken/TestController.sol

8a9c65aca396ef4e1026649b92d09ceb857fe5c35676519a9b9a3acbe7487fbe ./contracts/testToken/TestMintMaster.sol

75b72335744d61073fa09a381786472280f744c1f17fe492d7c4ce4b95319619 ./contracts/testToken/TestOracle.sol

5b2c4b42cc65ea902088233c19f44033afe174e2d56af0309e9489331ec10e56 ./contracts/testToken/Token18.sol

5613bbde40b67a5e86f423ed91bb2f91f2b7e19a28f461c6ac49f745583a6c1e ./contracts/testToken/Token6.sol

61d0a7fd3ebb6dd00cbff0caecd2c4ae0860902befa84e074b41eb494329ea88 ./contracts/testToken/Token9.sol

0867985495db1878ad2167c498f38334665424490cff00d3f0dace1018acc90c ./contracts/strategy/StrategyCommon.sol

900799a9a7a56b7697f6041bbddc12ba2564bd9edc1b4a4656e76ba0c82d4374 ./contracts/strategy/nullStrategy/NullStrategy.sol

63a63d16f0c3dd9729e4581e58cd45717ec120d37421c6e571c45afebbbd3b2f ./contracts/strategy/arbitrary/Arbitrary.sol

f32d72030bc94e456f8bce43bb93f9dcaefa84abbdf2a88a51a203c7b1968f82 ./contracts/oz_modified/ICHIERC20.sol

8fa310b8c82c2213240783df0da731fe3749eb50ace0c00b8a17fd39d0588b54 ./contracts/oz_modified/ICHIERC20Burnable.sol

bc4f667dc7ae446e49545c496fb8be546dbf6805fa3a69c1dddd09d5fd079301 ./contracts/oz_modified/ICHIInitializable.sol

1a23f7c19caee178b2a0952465567e505ddfe0cbfa10b16bdc7d61576a966dbb ./contracts/oz_modified/ICHIOwnable.sol

bd0d48c084294bd8ca8b2046dadf525f18fecf1719dd63eca90583ae9f702949 ./contracts/oracle/OracleCommon.sol

35467fcc6c88c790019aff0d107410e7834c89a3a1256643f4690cef9832a100 ./contracts/oracle/uniswap/UniswapOracleSimple.sol

00044e3d4ed7cf13236ee854f588c24e1e2552ec023c3445daacc41a72b82643 ./contracts/oracle/pegged/ICHIPeggedOracle.sol

66c66aa4ca23169eabfa21522d7367f0b36e39053f9d9d32dc9e4655e21f48ba ./contracts/oracle/composite/ICHICompositeOracle.sol

3e02d69e0c056c3151780c9303b4c80788ec7d7ecdcddd5ce0c3d2bbdd542e83 ./contracts/mintMaster/MintMasterCommon.sol

834ccdd73e2ed45f3704782c5fbf05f450e15ec9810ef81e97f3feb798d4f44b ./contracts/mintMaster/legacy/Incremental.sol

5e8a65b5a169d0c6d7cedc5fcb17c52d85ecaf8dc1a19d3e6cc1d5b9475263fa ./contracts/lib/AddressSet.sol

f2fd523702526d96450f1427f3b2fcc42486b43997d70efc65f75551a91a383f ./contracts/interface/IController.sol

95dd9a4cfe07445e1765b815c2aad927fcff47dcedf341903a2f4d01c5da1075 ./contracts/interface/IERC20Extended.sol

e42b6d07f96b373bc5b43160dd8c997e1ae2c7945c282b7f058f75ca05353b75 ./contracts/interface/IICHICommon.sol

37b3e45dd1032bc00ee803b1688fb56c1c458a18158a4d847a98d2858f5fd863 ./contracts/interface/IICHIOwnable.sol

5f1b5cace8310498b11a69ddab0dac585fd7c11dc98f91cf516bd2d8e0ab772f ./contracts/interface/IMintMaster.sol

368ff04e202dee4d59cadc1b18b9ac718f00363436fef0f8b1f79719e4e26f8f ./contracts/interface/IModule.sol

e2d3cff37adca855bcd442a4ececb6fb965223d3fcc0ca6fd176a6a75869e906 ./contracts/interface/InterfaceCommon.sol

5067f889d544c05bf51bee63ffdc23ec6b101058d85d6f45d000f1148823130a ./contracts/interface/IOneTokenFactory.sol

4a72418e17a0db5d89cae5a960f3fabece7914f6aa707c1b6cc62969cd3ca1a1 ./contracts/interface/IOneTokenV1.sol

81b799ea9ad1b82f2d4714186739d770e49bc679fc018fa2ae67ac8268e31469 ./contracts/interface/IOneTokenV1Base.sol

677677cc6442cf344aa1022cb1911809276731fc5ab8af969b822e2595feef8d ./contracts/interface/IOracle.sol

0f5c8c6deb4c94d48ad9232865a255556fc993cdf2956188ef14e527b44f4428 ./contracts/interface/IStrategy.sol

7e110814b3447e8077105d62c56d44fde07c126f55d5dfe3f15b5d0c07f1107f ./contracts/controller/ControllerCommon.sol

9a45b6fd779c5c74c244b7f78491708d9dd4f8e529274df939cedeb93267533f ./contracts/controller/null controller/NullController.sol

c7490510294342882e612ab15d01b6d216df22ade71e3f36e7396fc0106e8640 ./contracts/common/ICHICommon.sol

509f82b58d50507f07baf1507390aade7cab4f79453dd51d151bb5dbea0c81f3 ./contracts/common/ICHIModuleCommon.sol

Tests

61d34a442d187e9de699ecbd01c224841ae28a010b36d483293ddb61e176a118 ./test/addressSet.js

03a01e6eb783c8e834eca4d73ef6199276e32a1b6d08d0f825a45c4f44ea468a ./test/arbitraryStrategy.js

10912b52e2ba65b37e407dcf919ca5e570492e6e821b317c0d08b1bafa455769 ./test/controller.js

3f3ba62b85e2ff4e035583f518295dedd4926dfcadd1bddc427812b8b49cdfb9 ./test/factory.js

5fde1092d6db8d98efc93ff01fa6661f3783ddf398f1c73d2ee9dfa8b7ca59c3 ./test/ichiCompositeOracle.js

e4abda5b24f9852884318923410250 5e192fb85c0a819033a0d89d975a68034c ./test/ichiCompositeOracleDecimals.js

091ccdffc9ac701fa0414e005172d51306f71e8f35f6416681021b5d5158973e ./test/ichiPeggedOracle.js

883e7dc988531ae041aceffa37b011a87e0292d8469d23efb69b8e061356aa47 ./test/integration-exploratory.js

c0db345b7a198be8db2f9bfb999994d825b909a82660d56bd2e28fe5f9a2bb57 ./test/integration.js

3ed5634dc9d691177ef246c12bebdc4edd1429152733efb2f918a03160b58cbe ./test/integrationDecimals.js

2526e4c62733a4d2894961885d8f4409e3337d97948730706925e6e1565d4e0a ./test/mintMaster.js

e0aed6ed629133692ef545f41786c2e58941330bfa8812248158b2bf8679cad8 ./test/oneTokenV1-admin.js

79985ba055707f03806d275a3efdaab4e14cf55586ef73a0ad163fccd838ec9b ./test/oneTokenV1-base.js

b2cbd965244b15f1675557a388aa800a17ac403cea6dc92a5535a28f43d70d77 ./test/oneTokenV1-main.js

4d25d4d8f50f388cc0088f785deaa9c4ac07a9b6b1dc23a792be308c6bfc452f ./test/truffle-fixture.js

a959a63abd57ba7cc5d15fb801cf6369b1d34430b2e4af937e00f16f0e1514d8 ./test/uniswapOracleSimple.js

a17d566e18e42100a041b5c257146e2d1dc67d8cefcc0b2725332d8d6feda205 ./test/utilities/index.js

11c13f50163dbecca1082fca086ea10875f0409fab728289dc3e862b714e5d4d ./test/utilities/time.js

Changelog

- 2021-04-30 - Initial report
- 2021-05-24 - Revised report based on commit 42c2b7f

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.