Quantstamp Security Assessment Certificate

November 26th 2020 — Quantstamp Verified

Idle Governance

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

Executive Summary

Type	Yield Farming and Goverance
Auditors	Poming Lee, Research Engineer Kevin Feng, Blockchain Researcher Ed Zulkoski, Senior Security Engineer
Timeline	2020-10-12 through 2020-10-26
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md https://developers.idle.finance/

Source Code

Repository	Commit
idle-governance	91588bb
idle-governance	c2f5f04

Goals

- Do functions have proper access control logic?
- Are there centralized components of the system which users should be aware?
- Do the contracts adhere to best practices?

Total Issues	**8** (6 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**1** (1 Resolved)
Low Risk Issues	**2** (1 Resolved)
Informational Risk Issues	**4** (3 Resolved)
Undetermined Risk Issues	**1** (1 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 auditing, we found eight potential issues of various levels of severity: one medium-severity issue, two low-severity issues, four informational-level findings, and one undermined finding. We also made five best practices recommendations.
Disclaimer: Please be aware that Quantstamp was requested and had audited these files: `PriceOracle.sol`, `Idle.sol`, and `IdleController.sol`; the whole system was not audited by us.
update-2020-10-26: All findings were either fixed or acknowledged.

ID	Description	Severity	Status
QSP-1	Unintended Revert in Function `claimIdle`	Medium	Fixed
QSP-2	Gas Usage / `for` Loop Concerns	Low	Acknowledged
QSP-3	`_moveDelegates()` May Not Behave Correctly After Token Transfers	Low	Fixed
QSP-4	Missing Address Sanitization	Informational	Fixed
QSP-5	Privileged Roles	Informational	Acknowledged
QSP-6	`delegateBySig()` Should Validate the `v` and `s` Parameters	Informational	Fixed
QSP-7	Possible Truncation in Calculating APR Precision	Informational	Fixed
QSP-8	Integer Overflow / Underflow	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:

- Truffle v5.1.33
- SolidityCoverage v0.7.11
- Mythril v0.22.10
- Slither v0.6.12

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. Installed the Mythril tool from Pypi: `pip3 install mythril`
5. Ran the Mythril tool on each contract: `myth a path/to/contract`
6. Installed the Slither tool: `pip install slither-analyzer`
7. Run Slither from the project directory: `slither .`s

Findings

QSP-1 Unintended Revert in Function `claimIdle`

Severity: Medium Risk

Status: Fixed

Description: In `contracts\IdleController.sol`, the function `claimIdle` will loop through all the combinations of `holders` and `idleTokens` and pass into the function `distributeIdle`. However, in the function `distributeIdle` on `L119` will revert all the transactions that pass in any `supplier != idleToken`.

Recommendation: Remove the inner for loop in the function `claimIdle`, or remove `L119` directly. The way of fixing this issue should be done based on the functionality that the idle team actually seeks to achieve.

QSP-2 Gas Usage / `for` Loop Concerns

Severity: Low Risk

Status: Acknowledged

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 exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.
In particular, for `contracts/IdleController.sol`, if many markets are added, `refreshIdleSpeedsInternal()` may run into issues.
update-2020-10-26: Idle Team added a `_resetMarkets` method which should completely remove all markets in case of emergency. Gas analysis was also conducted successfully.

Recommendation: We recommend 1) performing gas analysis to ensure that each loop-function will not run into gas limitations, particularly for large inputs, and 2) adding a function that can reduce the number of markets in the `allMarkets` for emergency use.

QSP-3 `_moveDelegates()` May Not Behave Correctly After Token Transfers

Severity: Low Risk

Status: Fixed

Description: In `contracts\Idle.sol`, The function `_delegate()` invokes `_moveDelegates()` with the delegator's full balance instead of remaining undelegated balance. This can cause users to lose delegation ability if additional Idle tokens are acquired without minting (i.e., via transfers).
Consider the following scenario:

1. Alice has `10` Idle , which is delegated to Bob.
2. Alice acquires `1` additional Idle from a transfer.
3. If Alice attempts to re-delegate her `11` Idle tokens to Carol, it will fail due to the `SafeMath` check on `L195`; effectively, the function will attempt to undelegate `11` tokens from Bob instead of `10`, and revert.

In general, if Alice's balance is ever more than the number of tokens minted toward her account (due to transfers), she will not be able to delegate. This can be mitigated by Alice by simply transferring the excess tokens out of her account, however this scenario may not be clear to end-users from a UX-perspective.

Recommendation: It is not clear if this functionality is as intended. If so, no changes are needed, but user documentation should exist describing the scenario above. If the scenario above is undesirable, `_moveDelegates()` should be invoked in `_transfer()` as well. Note however that with this approach, votes can be more easily "bought" by acquiring Idle tokens on exchanges.

QSP-4 Missing Address Sanitization

Severity: Informational

Status: Fixed

Description: For `contracts\IdleController.sol`, the values inside the `priceOracle_` input parameter is not checked to be different from `0x0` inside the `_setPriceOracle` function.

Recommendation: Add a `require` statement that checks that the value of the `priceOracle_` is different from `0x0`.

QSP-5 Privileged Roles

Severity: Informational

Status: Acknowledged

Description: (a) For `contracts\PriceOracle.sol`, a potentially malicious owner (if the private key was leaked) can change the feed contract addresses/block length to give incorrect price oracles that can affect the token. While privileged roles for the Idle token are addressed in https://developers.idle.finance/advanced/admin-powers, it is not addressed for the price oracle. (b) For `contracts/IdleController.sol`, the admin can manipulate the rate of tokens in which each market receives by adding/removing idle markets to the list.
update-2020-10-26: Idle Team stated that the owner of the `PriceOracle.sol` and the `IdleController.sol` contracts will be the Timelock contract (i.e., the governance itself) directly on deploy.

Recommendation: These privileged operations and their potential consequences should be clearly communicated to (non-technical) end-users via publicly available documentation.

QSP-6 `delegateBySig()` Should Validate the `v` and `s` Parameters

Severity: *Informational*

Status: Fixed

Description: For `contracts\Idle.sol`, `delegateBySig()` should validate the `v` and `s` parameters as in `ECDSA.sol` (See: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/2bb06b1af4d57cf47c700992b327a08bebf64879/contracts/cryptography/ECDSA.sol#L46).

QSP-7 Possible Truncation in Calculating APR Precision

Severity: *Informational*

Status: Fixed

Description: For `contracts/PriceOracle.sol`, in function `getCompApr`, the computed pair will always return a result that has the last two digits as `0`, in `L58` due to division before multiplication `div(cTokenNAV).mul(100)`.

Recommendation: If it is not intended for the last two digits to be `0`, then it is recommended to perform `mul(100).div(cTokenNAV)` instead for a more precise calculation.

QSP-8 Integer Overflow / Underflow

Severity: *Undetermined*

Status: Fixed

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. Integer overflow and underflow may cause many unexpected kinds of behavior and was the core reason for the `batchOverflow` attack. Here's an example with `uint8` variables, meaning unsigned integers with a range of `0..255`. `function under_over_flow() public { uint8 num_players = 0; num_players = num_players - 1; // 0 - 1 now equals 255! if (num_players == 255) { emit LogUnderflow(); // underflow occurred } uint8 jackpot = 255; jackpot = jackpot + 1; // 255 + 1 now equals 0! if (jackpot == 0) { emit LogOverflow(); // overflow occurred } }`
In particular, for `contracts\PriceOracle.sol`, there is a potential underflow on `L70`: `... 10**(18-tokenDecimals) ...`. It is recommended to use `SafeMath` for this operation.

Automated Analyses

Mythril

Mythril reported no issues.

Slither

- Slither warns of several potential reentrancy issues, however as the associated external calls were to trusted contracts (either Idle contracts or underlying protocols), we classified these as false positives.
- Slither detects that there are "divided-before-multiplies" operations in `L58` of `contracts\PriceOracle.sol` as mentioned in our Finding section. Re-ordering these operations may improve precision.
- Slither detects that `comptrollerImplementation` in `contracts\IdleControllerStorage.sol` is never initialized as mentioned in our Best Practice section.

Adherence to Specification

The code adheres to the specification provided, as well as the inline documentation.

Code Documentation

The code is generally well-documented. We suggest several improvements:

- [false-positive] For `contracts\PriceOracle.sol`, in `L64` (and possibly `L67` also), the documentation says to `scale it to 1e18` but instead multiplies by `10e10`
- For `contracts\PriceOracle.sol`, the constant `100` in `L58` is not documented.
- Consider adding code comments to the functions that currently are not commented, to increase the maintainability of the code.

Adherence to Best Practices

The code does not fully adhere to best practices. In particular:

- For `contracts\PriceOracle.sol`, consider using constants for the key part of the assignment for `L28~L29` and `L32~L37` like what is done in `L26` and `L27`, in order to make the code more readable.
- [false-positive] `L48` in `contracts\IdleController.sol`, `comptrollerImplementation` was never set.
- For `contracts\PriceOracle.sol`, the ChainLink price feeds are hardcoded in the constructor, for better coding practices it is better to pass this in as an `initial value` during the deployment of the contract.
- For `contracts\Idle.sol`, should use `uint256` instead of `uint`.
- [false-positive] For `contracts\IdleController.sol`, Comp has the public convenience function `claimComp(address holder)` which claims for all markets. Is there a reason this was omitted?

Test Results

Test Suite Results

All tests have passed.

```
Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: EarlyRewards
     ✓ set params on constructor (74ms)
     ✓ allows onlyOwner to setRewards (103ms)
     ✓ setRewards reverts if params have different length
     ✓ setRewards reverts if canSetReward is false (64ms)
     ✓ recipient can claim (104ms)
     ✓ claim reverts if msg.sender is not authorized (108ms)
     ✓ emergencyWithdrawal can only be called by owner can claim
     ✓ emergencyWithdrawal can be called by owner for any non IDLE token (75ms)
     ✓ emergencyWithdrawal can be called by owner for IDLE token with any address but fund goes to ecosystem (76ms)

  Contract: IdleController
     ✓ refreshIdleSpeeds with IDLE equally splitted (150ms)
     ✓ refreshIdleSpeeds with different supply (256ms)
     ✓ refreshIdleSpeeds with different price (240ms)
     ✓ refreshIdleSpeeds with different aprs (239ms)
     ✓ refreshIdleSpeeds with different asset prices (287ms)
     ✓ refreshIdleSpeeds with tokens with different decimals (308ms)
     ✓ _resetMarkets (269ms)

  Contract: PriceOracle
     ✓ allows onlyOwner to setBlocksPerYear (62ms)
     ✓ allows onlyOwner to updateFeedETH (63ms)
     ✓ allows onlyOwner to updateFeedETH (61ms)
     ✓ getPriceUSD when there no price feed
     ✓ getPriceUSD when there is USD price feed (64ms)
     ✓ getPriceUSD when there no USD price feed but ETH price feed is present (72ms)
     ✓ getPriceETH when there no price feed
     ✓ getPriceETH when there is ETH price feed (40ms)
     ✓ getPriceToken when there no price feed for asset
     ✓ getPriceToken when there no price feed for token
     ✓ getPriceToken when there are both price feeds (103ms)
     ✓ getUnderlyingPrice (41ms)
     ✓ getCompApr (88ms)


  29 passing (18s)
```

Code Coverage

The branch, statement and function coverage of `contracts\Idle.sol` and `contracts\IdleController.sol` are low. This indicates that much of the functionality of the protocol is not executed during tests. We strongly recommend that the branch coverage be brought to 100% as it is crucial to execute all functionality in order to verify that no functional bugs exist in the code.

update-2020-10-26: Idle Team states that they have added only tests for differences with respect to the audited version of the code of Compound. They is why for `Idle.sol` and `IdleController.sol` tests, given that they had the majority of the code copied from Compound, don't have that much coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	27.18	15.2	29.25	27.47	
ERC20Permit.sol	0	0	0	0	… 63,67,68,70
EarlyRewards.sol	100	83.33	100	100	
EcosystemFund.sol	0	100	0	0	11
GovernorAlpha.sol	0	0	0	0	… 323,324,325
Idle.sol	0	0	0	0	… 267,271,272
IdleController.sol	66.94	37.04	70	66.94	… 273,274,310
IdleControllerStorage.sol	100	100	100	100	
PriceOracle.sol	100	100	100	100	
Reservoir.sol	0	0	0	0	… 90,94,95,97
Timelock.sol	0	0	0	0	… 108,110,115
Unitroller.sol	0	0	0	0	… 126,136,138
Vester.sol	0	0	0	0	… 53,55,60,62
VesterFactory.sol	0	0	0	0	… 50,51,53,57
contracts/interfaces/	100	100	100	100	
CERC20.sol	100	100	100	100	
ChainLinkOracle.sol	100	100	100	100	
Comptroller.sol	100	100	100	100	
IdleToken.sol	100	100	100	100	
contracts/lib/	18.75	12	26	18.75	
CarefulMath.sol	0	0	0	0	… 77,79,80,83
Exponential.sol	22.83	15.79	28.89	22.83	… 331,335,348
contracts/mocks/	90.48	100	90	90.48	
CERC20Mock.sol	71.43	100	60	71.43	18,21
ChainLinkOracleMock.sol	100	100	100	100	
ComptrollerMock.sol	100	100	100	100	
IdleTokenMock.sol	100	100	100	100	
PriceOracleMock.sol	100	100	100	100	
Token8Mock.sol	100	100	100	100	
TokenMock.sol	100	100	100	100	
All files	**27.79**	**14.67**	**35.23**	**28.03**	

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

```
22f1b43d3d1a61e7a4435ab0d7b9ad79200500a5489d92a4b5ff397ba1712e07 ./contracts/EarlyRewards.sol
98e2c874a8edcd58e3767a371b008e159a44940431fe0a7942e38388ba4efd36 ./contracts/EcosystemFund.sol
252882099f640ca0a522cf0ee298cdc2f299fd04859c4c1dd36633cd8db5b693 ./contracts/ERC20Permit.sol
7aecd1127ea8f5f4d8f48eb80a2745b9f47069a3eb84d860f9957e6529cdb413 ./contracts/GovernorAlpha.sol
bf7254423db86002329365c59fcfa08b80a42ab00d30049a86adbecfcc0bf71f ./contracts/Idle.sol
69353b0ff3ec9da8c5982ff28673975022e1d6fb23176cf1a5146eb4d65a9156 ./contracts/IdleController.sol
cf1bb5be11d6dbd7e6882fe2e8e039decf389038ef4f15d79f5aeed872653964 ./contracts/IdleControllerStorage.sol
af517e1c7773edd612f148b79be5f99b710770d16f3b90a59e79c0c438a622b2 ./contracts/Migrations.sol
523823761ff2517903c1aaf4f8d240ab6cd36d2a9e972b76e98ab1182088fd72 ./contracts/PriceOracle.sol
fd89fbde26cceba50b2c8ff960195ebc682fa50025a3cc1c0a062944116e9392 ./contracts/Reservoir.sol
d96fd3db5dcb78fa62d6c3c62c0cbc69af20fbec82c3714960d6569c1a0c6ec1 ./contracts/Timelock.sol
0d1cb7994cc926a592d107d7d2577eab450d6d581da06a04b4580e9a72a652d7 ./contracts/Unitroller.sol
9438313d1a2422d6bc492a4f3fed2cf47d24d4f6b4a01dbaf58be142f1898634 ./contracts/Vester.sol
9096d9e048ff2fe1146a64d40ad78bc768f5bb930787d88c4503dd0ae14291a1 ./contracts/VesterFactory.sol
59f28a92602fc8db4a19fdc20b1f6668276cfce32c9a5007a878d292eb708429 ./contracts/mocks/CERC20Mock.sol
799e038edda5225839238f7f09e1c32c129fb1ceefb238106f520131634868ff ./contracts/mocks/ChainLinkOracleMock.sol
5f4b7719cd71ad190a85b10eb919f3915ea5d713e0c6308c2d929c0ab532ae99 ./contracts/mocks/ComptrollerMock.sol
efcaaa15b737386149a1b36e21cf013fe64bc09f9070e30f5b327d29704780a5 ./contracts/mocks/IdleTokenMock.sol
5fc682a8d24d2cb3c5e3ad65874cbe17104fd311bca56ab65e2449681d48629f ./contracts/mocks/PriceOracleMock.sol
418f700b45778517311e4ee3c2541706c67dd01abe045277ab90d03a596c0de4 ./contracts/mocks/Token8Mock.sol
757c658b1f4702da6eeaead720d761e3a47ec853eedf80def11a1e498123f1ab ./contracts/mocks/TokenMock.sol
70e97f49e0a6b17852f911a28b6ca3483c58ba016b39c0cee2669cb57ba01934 ./contracts/lib/CarefulMath.sol
1317006b25a9a3db0617687ba229377a2401343ed42286b09b1b92e2c1199b8c ./contracts/lib/Exponential.sol
2afe580ca19a58668ac7d2c921e41871ddf5be5acbff53aad675fdbb4dc0e9ae ./contracts/interfaces/CERC20.sol
6583d26353cc86749627494c75cec775ca64c257a5920195a63f3a10529f9e11 ./contracts/interfaces/ChainLinkOracle.sol
04867dae08479e7c182c2bbe2d88272796171b208a2cbd7b52d1cfd57e8f0d07 ./contracts/interfaces/Comptroller.sol
1e41d4dd2cb51a14b0f9ee6515e160cdc69e6a7192b20233456aea124fb42dc8 ./contracts/interfaces/IdleToken.sol
```

Tests

```
feacb25facac1f080eb55b3f73d10ae3312c58af922d10a0d917806a57b4c9f1 ./test/EarlyRewards.js
a746ffe36cfe4b9e66313787c8ff7568d9868c3c382453f818996ddc8191f320 ./test/IdleController.js
46009cac4439f8775be0d59e28d59907c9b6748c90874d2da9170610fe41a6f1 ./test/PriceOracle.js
```

Changelog

- 2020-10-20 - Initial report
- 2020-10-26 - Reaudit report

About Quantstamp

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

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.