Quantstamp Security Assessment Certificate

October 29th 2020 — Quantstamp Verified

Auric Network

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	Token and distribution platform
Auditors	Martin Derka, Senior Research Engineer Joseph Xu, Technical R&D Advisor Sung-Shine Lee, Research Engineer
Timeline	2020-10-12 through 2020-10-29
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README
Documentation Quality	Medium
Test Quality	Medium
Source Code	Repository: auric-finance; Commit: 177b244

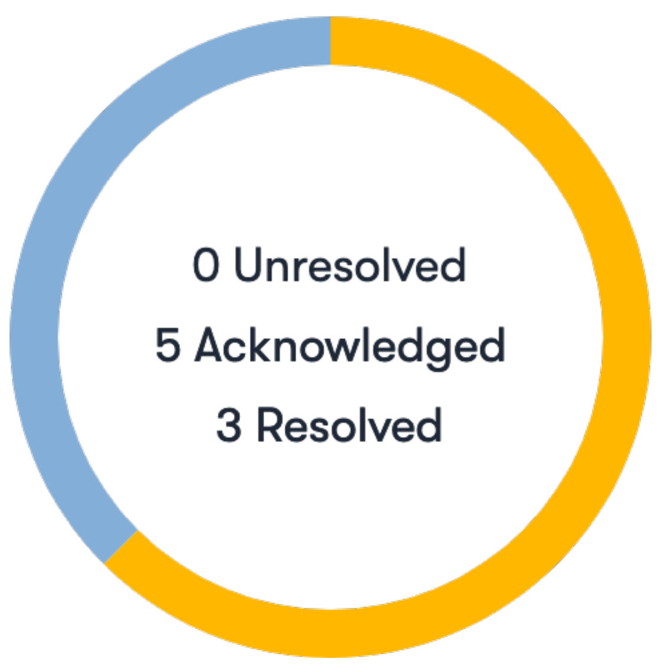
Source Code

Repository	Commit
auric-finance	177b244

Goals

- To assess security of the token
- To assess the security of the distribution platform
- To assess the security of the user funds inside the distribution platform

Total Issues	**8** (3 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**2** (1 Resolved)
Informational Risk Issues	**5** (2 Resolved)
Undetermined Risk Issues	**1** (0 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The platform of Auric Network consists of an ERC20 token with governing and rebasing functionality (in a great extent cloned from the Compound and Yam project---both previously audited), and the token distribution platform via staking pools by SNX. The token distribution platform contains code additions to support the rebasing token. The code of the token is simplified by removing some functionality.

The security of some parts of the smart contracts rely on on-chain configuration. Quantstamp was notified that the token and the distribution platform, less rebasing, timelock, and the governor, are already deployed on mainnet, and thus is able to verify the present configuration. Quantstamp currently deems the distribution mechanism safe, with the exception of the listed findings. The notoriously known SIP-77 issue is fixed in the code. The auditing team was able to confirm that the deployer of the contracts does not have the ability to remove the funds from staking pool. As outlined, a configuration error could lead to disabling the reward claiming (and consequently withdrawing functionality), which is currently not the case, and the Auric Network team confirmed that no re-configuration is intended in the future. The Quantstamp team also confirmed that the token respects the ERC20 interface and does not contain overflow errors. A misconfiguration of the rebasing functionality can lead to the denial of service on the token contract, however, the configuration can be controlled by the governance which can correct such actions. The codebase contains some privileged features that are intended to be controlled by the governance.

The codebase overall appears free of defects, with the exception of the findings outlined in the report. Quantstamp did not discover an option of the contract deployer being able to execute a so-called "rug pull" or otherwise access the staked funds when the platform is deployed and correctly configured on chain.

ID	Description	Severity	Status
QSP-1	Rebase During Warm-Up Phase	Low	Fixed
QSP-2	Initialization Race Pattern	Low	Acknowledged
QSP-3	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-4	Unenforceable Interfaces	Informational	Acknowledged
QSP-5	Privileged Roles and Ownership	Informational	Acknowledged
QSP-6	Oracle Dependency	Informational	Acknowledged
QSP-7	Strict Inequality	Informational	Fixed
QSP-8	Low Quorum Requirement	? Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

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

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

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Toolset

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

Setup

Tool Setup:

- Slither v0.6.6

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .s`

Findings

QSP-1 Rebase During Warm-Up Phase

Severity: Low Risk

Status: Fixed

Description: The `rebase()` function can be called during the warm-up phase for generating average prices if `setNextRebase()` is not set properly.

Recommendation: Quantstamp recommends adding a check `counter > WINDOW_SIZE` to ensure that the desired number of price reads was reached.

QSP-2 Initialization Race Pattern

Severity: Low Risk

Status: Acknowledged

Description: The platform uses the initialization pattern to set some parameters, instead of those being placed in the constructor. This includes especially `setPendingAdmin()` in `TimeLock.sol`. Similarly, anyone can call `rebase()` on `Rebaser.sol` if `nextRebase` timestamp is set below current `block.timestamp`. Such an undesired rebase could be out of schedule, and would require re-deploying and configuring new rebaser to AUSCM as the current `nextRebase` timestamp would be wrong (potentially far in the past).
It is imperative that successful parameter initialization is confirmed before the contracts are set to their expected roles.

Recommendation: Quantstamp recommends verifying successful contract initializations during deployment.

Update: The Auric team acknowledged the design and understands the necessity of configuration ordering and proper deployment.

QSP-3 Allowance Double-Spend Exploit

Severity: Informational

Status: Mitigated

File(s) affected: `AUSC.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens. An example of an exploit goes as follows:

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

Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

Recommendation: The issue is already mitigated via the presence of methods `increaseAllowance()` and `decreaseAllowance()`. Quantstamp recommends for the Auric team to communicate the presence of these methods to the users.

QSP-4 Unenforceable Interfaces

Severity: Informational

Status: Acknowledged

Description: The code orchestrates several smart contracts making calls to each other. If the platform is deployed in a way that expected interfaces become unavailable, transactions may start failing. This includes especially the following:

- The AuricRewards contract makes a call to PoolEscrow to release AUSC tokens. The required interface has to be implemented by the configured escrow contract.
- The PoolEscrow makes a notification to the secondary pool escrow whenever tokens are being released. The required interface has to be implemented in the configured contract, unless it is the `0x0` address.
- The AUSCM token itself makes calls to the Rebaser upon any transfer. The required interface has to be implemented by the rebaser if it is configured, otherwise the transaction will fail.
- The governance has to be able to make calls to the token and other contracts that it governs, otherwise new configurations may become impossible.

Recommendation: A portion of the platform is already deployed, and Quantstamp verified that the required interfaces are implemented. Given this fact, the vulnerability is assessed as informational, otherwise its classification could be more severe as it could have an DoS impact on the platform. Quantstamp recommends documenting the interfaces and informing the community about the configuration sensitivity when enforcing the governance.

QSP-5 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner. In the case of the Auric platform, the privileged access is intended to be held by the on-chain governance contracts, but it is important to realize that the platform may not be configured this way post-deployment, and that the holder of this role can change.
The privileged access allows:

- Manipulating the AUSCMs in the escrows.
- Rebasing and minting AUSCMs.
- Configuring the secondary reward pools, and escrows that receive partial rewards from the pool escrows.
- Configuring the governance.

Recommendation: Quantstamp recommends informing the community about the options of the privileged users and the governance.

QSP-6 Oracle Dependency

Severity: *Informational*

Status: Acknowledged

Description: The platform depends on external oracles for the purpose of determining the price of gold (Chainlink) and AUSCM (Uniswap). The price that the rebasing decision is based on is an average over 12 reads spread at least by one hour. If the oracles become unavailable, the price will not be recorded. If the oracles report a dishonest price, the read will record a dishonest price, and the token can rebase to a false target. (Note that the platform includes a protection against on-chain market manipulation via smart contracts, e.g. by using using flash loans, so this finding refers to a true ingenuity of the oracle.)

Recommendation: The vulnerability appears to be inherent to the design of all rebasing tokens. Furthermore, a temporary diversion from the price target can be corrected in the next rebase. Quantstamp recommends informing the community about this dependency, and monitoring the accuracy of the oracle.

QSP-7 Strict Inequality

Severity: *Informational*

Status: Fixed

File(s) affected: `GovernorAlpha.sol`

Description: There is a minor mismatch with spec in that the implementation requires the proposer to exceed 1% of the AUSC supply to be able to propose (Line 170) as opposed to "at least 1% of all the tokens" as described.

Recommendation: Quantstamp recommends removing the bug.

QSP-8 Low Quorum Requirement

Severity: *Undetermined*

Status: Acknowledged

Description: Any proposal will automatically reach quorum since `quorumVotes()` returns `proposalThreshold()`. If the proposer votes "for", then the default state of any proposal is to pass. The docs and code (Line 33) all mention this feature but it might be worth making extra clear since this mechanism is quite different from the behavior of Compound, YAM, or other standard governance usages.

Recommendation: Quantstamp recommends informing the users about the quorum.
Response from Auric: This setting is intended. Several platforms in the past days had problems with reaching quorum on-chain while attempting to decrease quorum requirements that were set too high. Therefore, we choose to start with a low quorum, accompanied with a sufficiently long voting period for everyone to be able to express their opinion, and a time lock on approved transactions that is long enough for users to exit if they disagree with the vote, and leave it up to the governance to increase the quorum through a proposal if this is a desired feature.

Automated Analyses

Slither

Slither failed to resolve certain import paths and did not succeed in analyzing all the contracts. All reported issues (multiplication after division, re-entrancies from the rebaser, public instead of external declarations) were confirmed as benign.

Adherence to Specification

The code adheres to provided specification.

Code Documentation

The documentation is concise and to the point. The ideas and desired goals appear to be clear.

Adherence to Best Practices

- `AUSC.sol`: Incorrect in-line comments in the `_burn()` function lines 129 and 136.

A few in-line comments were not updated from the YAM contracts:

- Line 42 in `./contracts/token/TokenStorage.sol`

- Line 123 in `./contracts/token/Governance.sol`

Update: The highlighted best practice issues were addressed.

Test Results

Test Suite Results

The tests appear to focus on the functional side of the code (initialization of the platform, token transfers, distribution, rebasing, and voting). The custom code appears tested reasonably well, but some of the clone is not cloned with tests. Quantstamp recommends adding tests for all uncovered functions.

```
Contract: AUSC Test
  Basic initialization and other functions
     ✓ symbol and decimals are correct after initialization (393ms)
     ✓ ERC20 functions (1131ms)
     ✓ Minting and burning (213ms)
     ✓ token rescue by governance (422ms)
     ✓ changing governance (444ms)
  Rebasing
     ✓ setting monetary policy
     ✓ positive rebase (1334ms)
     ✓ positive real rebase (670ms)
     ✓ positive real rebase 2 (843ms)
     ✓ negative rebase (513ms)
     ✓ no rebase (196ms)
  Governance
     ✓ votes (979ms)
     ✓ complex governace (4001ms)

Contract: Monetary Policy Test
  Average computation test
     ✓ running average constant, no time waiting (1795ms)
     ✓ running average constant, with time waiting (3643ms)
     ✓ moving the window, with time waiting (2297ms)

Contract: Rewards Test
  Basic initialization
     ✓ pool escrow setters (1028ms)
     ✓ set rewards distribution (72ms)
     ✓ set escrow (259ms)
     ✓ transfer and notify (2444ms)
     ✓ double notify (2450ms)


21 passing (55s)
```

Code Coverage

The test coverage overall appears low. Most of the uncovered lines and branches appear to be code clones from other repositories. The custom features of the codebase appear covered well.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	64.63	53.75	64.15	65.75	
AUSC.sol	98.95	71.43	100	99	384
ApyOracle.sol	0	0	0	0	… 91,92,94,95
BasicRebaser.sol	73.86	67.65	66.67	75.56	… 213,214,216
ChainlinkOracle.sol	0	100	0	0	20,22
Escrow.sol	0	0	0	0	… 23,24,28,32
IAUSC.sol	100	100	100	100	
IPoolEscrow.sol	100	100	100	100	
IRebaser.sol	100	100	100	100	
Rebaser.sol	100	100	0	100	
Rewarder.sol	0	0	0	0	… 18,19,22,23
UniswapOracle.sol	0	100	0	0	22,23,28,29
contracts/distribution/	85.71	53.23	78.69	85.62	
EscrowToken.sol	100	100	100	100	
SecondaryEscrowToken.sol	100	100	100	100	
SnxPool.sol	85.62	53.23	77.97	85.53	… 540,544,545
contracts/governance/	74.45	43.48	67.74	73.23	
GovernorAlpha.sol	74.19	46.55	68.18	72.29	… 395,404,405
TimeLock.sol	75	38.24	66.67	75	… 139,141,172
contracts/mocks/	100	100	100	100	
MockGovernorAlpha.sol	100	100	100	100	
MockOracle.sol	100	100	100	100	
MockRebaser.sol	100	100	100	100	
TestToken.sol	100	100	100	100	
contracts/token/	70.21	63.64	87.5	71.74	
Governance.sol	70.21	63.64	87.5	71.74	… 113,116,162
GovernanceStorage.sol	100	100	100	100	
TokenInterface.sol	100	100	100	100	
TokenStorage.sol	100	100	100	100	
All files	**73.02**	**50.78**	**73.46**	**73.29**	

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

```
68695caad96c91e68e439cf6f58e01b3c5d6c33962db55325fe2a1ad6e4fc792 ./contracts/IRebaser.sol
f3c5f94b558dc3babeda702eb59a0177a779e5c740420b1ffdfffb3294a9c583 ./contracts/IPoolEscrow.sol
c7181e766a11f52274bfd21420955789dcdd01cc24ab34f16ab7f36a0d9776d5 ./contracts/UniswapOracle.sol
eb66e81050aa065f49854047159641a15d78cbc3a5a7a115d74b5a834cec32f4 ./contracts/Migrations.sol
f5836b4328b86e324d49a10e3c87688384096fb9cc5849a552d58789916d2a1f ./contracts/BasicRebaser.sol
17767faae2e8d475ffe32511bd369965dafb9e81ae5c4e1e6d7fbca44625088b ./contracts/IAUSC.sol
843ad66d8275dd0c314c8615910fa6b01f7bde0ba6f0d3de5f5e026b310772fb ./contracts/Escrow.sol
0c2db6b3af5c12fc649a124779a5e52aff7063076876ce916ce5cf0a78f8d747 ./contracts/Rebaser.sol
4f99c011be59af45bf267b651f782133e56a7660bcea6c25b1d81fae655ad1c5 ./contracts/Rewarder.sol
ca0c4df28bef8c0597b8d966d65e48fbc5cb858f2b0bf775a71592b0d23511fd ./contracts/AUSC.sol
9cb589f8b904e1d6bd5466bc6f914082e271fcd1efdcb996587bba18473544ae ./contracts/ChainlinkOracle.sol
d137a393d731381ae80fd6d7c20d1f70fd34c94036be817a9a640d50834281ce ./contracts/ApyOracle.sol
b35f02e3fceab99aed8ff0fe6aaa4e5cf90a6aa0bf541381520c392849298247 ./contracts/governance/GovernorAlpha.sol
ac5892aa5896b5dca8ddb92a825b64152e3d38980340c9653a55758f1de18b7b ./contracts/governance/TimeLock.sol
ea556c3224cb975da101025af1f4f6094d2c29494c050821d502bc5f2ada3717 ./contracts/distribution/EscrowToken.sol
11185d592023fa7642497900e2e7e7b76eae997ef53bb4f7a8af2d61a37e4ac6 ./contracts/distribution/SnxPool.sol
2bf667d939dd30ad8408d7acf6fc2c58db593f2ef9a66c63f9d9eb98b1abcb38 ./contracts/distribution/SecondaryEscrowToken.sol
48d289a7bf0bcd35843ac3890b1c4bb8f2018b99deac11d4a1a0ba1c0338a22d ./contracts/token/TokenStorage.sol
3a00110a6ba0cfd8c3f68e174c673eafe1b5172892e4b1e81f4c382c65b4749e ./contracts/token/Governance.sol
6c1a8be861e811ec2d71268a7881af6a71551a48f05e1b9397a522a73535eb2a ./contracts/token/GovernanceStorage.sol
3a638e14339e48505a2186b639719fbef707e35fb1aedf7cf4f60c03b5499d21 ./contracts/token/TokenInterface.sol
0691d614ebf87e7b07e78ec20a3a9830d92b2dce99e513dda1206ffd87546cb7 ./contracts/mocks/MockRebaser.sol
8a415b97e6d3d176981a23722bde1711028144c60de89735cb62f6fbe0971324 ./contracts/mocks/MockGovernorAlpha.sol
235b98334f489caa55f74bbfe544a1fda38c0184b6a66a7e6702b93411e40a6d ./contracts/mocks/MockOracle.sol
0130fa01feb1c413603df0e98a3e340a93394b4102d2768354228d86fbf5e321 ./contracts/mocks/TestToken.sol
```

Tests

```
3724f9f0f84fa66c21a3feb62babf7c0f51532859c0b16c4e40a9f5e3d4dd9c9 ./test/ausc.js
6a1024b1bbf63eed2d06cb8c7aa225def5d48f37cda1150aeab70646cd869ef5 ./test/rebaser.js
32c898c5ad586fbc5a8f14ff213cc322674e1322bc05caa20b0b0dc18413893d ./test/snx.js
```

Changelog

- 2020-10-26 - Preliminary in-progress report
- 2020-10-28 - Completed report delivered to the client
- 2020-10-29 - Final report based on 06347dd7

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.