Quantstamp Security Assessment Certificate

March 30th 2021 — Quantstamp Verified

DAOFi

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

Executive Summary

Type	Decentralized exchange
Auditors	Kacper Bąk, Senior Research Engineer Fayçal Lalidji, Security Auditor
Timeline	2021-02-02 through 2021-02-24
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review
Specification	DAOfi Whitepaper
Documentation Quality	High
Test Quality	Medium

Source Code

Repository	Commit
daofi-v1-core	69e5b8d
daofi-v1-periphery	03081ef

Goals

- Can funds get locked up in the contract?
- Are computations implemented correctly?
- Can unauthorized user withdraw funds?

Total Issues	**15** (6 Resolved)
High Risk Issues	**4** (4 Resolved)
Medium Risk Issues	**2** (0 Resolved)
Low Risk Issues	**5** (2 Resolved)
Informational Risk Issues	**4** (0 Resolved)
Undetermined Risk Issues	0 (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

We have found quite a few vulnerabilities in the code. They span all severity levels. Notably, there are 3 high-severity, and 2 medium-severity issues. Futhermore, although project has proper documentation, specification for its logic is missing. We highly recommend addressing all the issues before deploying the code.
Update: the team addressed all of our findings.

ID	Description	Severity	Status
QSP-1	Incorrect Conversion	High	Fixed
QSP-2	Poorly documented logic	High	Fixed
QSP-3	Swap at fixed price	High	Fixed
QSP-4	Unintentional swaps	High	Fixed
QSP-5	Denial-of-Service (DoS)	Medium	Acknowledged
QSP-6	Incorrect input validation	Medium	Acknowledged
QSP-7	Input Validation	Low	Fixed
QSP-8	Numerical precision	Low	Acknowledged
QSP-9	Privileged Roles and Ownership	Low	Acknowledged
QSP-10	Race Conditions / Front-Running	Low	Fixed
QSP-11	Anyone Can Provide Liquidity	Low	Acknowledged
QSP-12	Gas Usage / `for` Loop Concerns	Informational	Acknowledged
QSP-13	Rebasing tokens	Informational	Acknowledged
QSP-14	Code hash computation	Informational	Acknowledged
QSP-15	Add and remove liquidity design	Informational	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.

Findings

QSP-1 Incorrect Conversion

Severity: *High Risk*

Status: Fixed

File(s) affected: `DAOfoV1Pair.sol`

Description: In the function `DAOfoV1Pair._convert()`, when dividing `amount` by `factor`, if `amount` is lower than factor, the input `amount` is not converted (probably to avoid truncation). The function will return the unconverted amount without any information regarding if it was converted or not. All implemented functions that use `_convert()` will process the returned value as if it was converted which is incorrect and can lead to loss of funds for both liquidity provider and users.

Recommendation: We recommend verifying this behavior and checking what value should be returned (either the original value or 0). Also, consider adding another return value that informs whether the conversion succeeded or not.
Update: the issue appears fixed but the internal decimal value is set to 8 meaning that the truncation for tokens with 18 decimals will be 10 decimals which is quite high. To avoid truncation, the project could use larger number of decimals.

QSP-2 Poorly documented logic

Severity: *High Risk*

Status: Fixed

Description: The pool logic is poorly documented in the code, which impairs external auditing. Furthermore, the implemented logic lacks proper specification. For instance, we couldn't determine the severity of some of the issues and their impact on the system.

Recommendation: We recommend documenting the code carefully to communicate the logic clearly.

QSP-3 Swap at fixed price

Severity: *High Risk*

Status: Fixed

File(s) affected: `DAOfiV1Pair.sol`

Description: In the function `getQuoteOut()` if `amountBaseIn` is equal to or higher than `supply`, the function returns the value of `amountQuoteOut` equal to `quoteReserve` meaning that the token swap is be calculated without any slippage (fixed exchange rate) which is contradictory to the implemented bonding curve where the price varies following the amount to be exchanged. Furthermore, user could exchange too many input tokens, since the function returns the same amount out for any value `amountBaseIn` >= `supply`.

Recommendation: We recommend verifying whether this is the intended behavior.
Update: we marked the issue as fixed assuming that `saleTargetAmount()` works as intended. The project, however, relies on an outdated Bancor formula function. We recommend checking whether the formula itself needs to be updated.

QSP-4 Unintentional swaps

Severity: *High Risk*

Status: Fixed

File(s) affected: `DAOfiV1Router01.sol`

Description: Unintentional swaps may occur in the functions `swapExactTokensForTokens()` and `swapExactTokensForETH ()` since both take `sp.sender` as an argument instead of using `msg.sender`.

Recommendation: Unless there is a good reason to keep the current design, we recommend using `msg.sender` instead of `sp.sender`.

QSP-5 Denial-of-Service (DoS)

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `DAOfiV1Factory.sol`, `DAOfiV1Router01.sol`

Description: A Denial-of-Service (DoS) attack is a situation which an attacker renders some functionality unavailable. Specifically the function `createPair()` is prone to a DoS attack through front-running. A malicious user may front-run the function and provide a different `pairOwner` argument. Consequently, a new pair contract would have an owner chosen by the attacker. The same issue applies to functions `addLiquidity()` and `addLiquidityETH()`.

Recommendation: We recommend baking in constructor arguments into the code that gets deployed via `createPair()`.

QSP-6 Incorrect input validation

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `DAOfiV1Pair.sol`

Description: When initializing `DAOfiV1Pair` using the function `initialize()`, following the natspec documentation, `_n` value should be between 1 and 3. However, the hardcoded `MAX_N` value is equal to 1 instead of 3 as documented.

Recommendation: We recommend adjusting the code or documentation.

QSP-7 Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `DAOfiV1Factory.sol`, `DAOfiV1Router01.sol`, `DAOfiV1Pair.sol`

Description: The following functions do not check if arguments of type address are non-zero:

- `DAOfiV1Pair.initialize()`,
- `DAOfiV1Pair.setPairOwner()`,
- `DAOfiV1Factory.constructor()`,
- `DAOfiV1Factory.createPair()`,
- `DAOfiV1Router01.constructor()`.

Recommendation: We recommend adding the relevant checks.

QSP-8 Numerical precision

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `DAOfiV1Pair.sol`

Description: The function `_convert()` converts token amounts so that they have 8 decimals. This may result in lost precision.

Recommendation: We have no further recommendations since the team acknowledged the issue.

QSP-9 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `DAOfiV1Pair.sol`

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

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.

QSP-10 Race Conditions / Front-Running

Severity: *Low Risk*

Status: Fixed

File(s) affected: `DAOfiV1Pair.sol`

Description: A block is an ordered collection of transactions from all around the network. It's possible for the ordering of these transactions to manipulate the end result of a block. A miner attacker can take advantage of this by generating and moving transactions in a way that benefits themselves.
Specifically, there may be a race condition between transferring tokens to the pair contract and calling the function `swap()`.

Exploit Scenario:

1. User A transfers `tokenIn` tokens to the pair contract in transaction T1.
2. User B calls `swap()` passing the same `tokenIn` in transaction T2. It gets executed right after T1 and User B swaps the previously transferred user A's tokens.
3. User A calls `swap()` (after T2) but the transaction fails since all the tokens have already been exchanged.
4. User A loses tokens sent to the pair contract.

Recommendation: We recommend one of the following: 1) informing users about this behavior and asking them to use the router contract, 2) redesigning the contract so that there is some accounting regarding who sent how many tokens, 3) restricting the contract so that only router can use it.

QSP-11 Anyone Can Provide Liquidity

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `DAOfiV1Router01.sol`

Description: Since the pool is designed to allow only one liquidity provider, only the owner should be able to add and remove liquidity via `addLiquidity()` and `removeLiquidity()`.
If another user uses a deployed pair and provides liquidity (through `addLiquidity()` or `addLiquidityETH()`), they will not be able to withdraw their deposit since only the owner is allowed to do so. Furthermore, a malicious user can deposit a small amount of tokens to deny service for the pool owner.

Recommendation: We recommend adding checks so that only the pool owner can add and remove liquidity.

QSP-12 Gas Usage / `for` Loop Concerns

Severity: *Informational*

Status: Acknowledged

File(s) affected: `DAOfiV1Pair.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. Specifically, unvetted tokens may consume all the passed gas in L84.

Recommendation: We recommend carefully checking the tokens before using them in DAOfi.

QSP-13 Rebasing tokens

Severity: *Informational*

Status: Acknowledged

File(s) affected: `DAOfiV1Pair.sol`

Description: DAOfi pair contract computations are incompatible with rebasing tokens.

Recommendation: We recommend against using DAOfi with rebasing tokens.

QSP-14 Code hash computation

Severity: *Informational*

Status: Acknowledged

File(s) affected: `DAOfiV1Library.sol`

Description: The L25 contains hardcoded code hash.

Recommendation: Instead of hardcoding the bytecode hash in `DAOfiV1Libary`, import `DAOfiV1Pair` contract and calculate the bytecode hash inside the contract to avoid future errors.

QSP-15 Add and remove liquidity design

Severity: *Informational*

Status: Acknowledged

File(s) affected: `DAOfiV1Pair.sol`

Description: The functions `deposit()` and `withdraw()` can be called only once by the router contract, meaning that users won't be able to provide liquidity after the initial deposit call. Such implementation will cause higher slippage since there will be less liquidity in the pools (only one user is allowed to deposit, and only the owner is allowed to withdraw).

Recommendation: The team should clearly define if this is the intended behavior.

Adherence to Best Practices

1. in `DAOfiV1Pair.sol#327`, there is no need to check for the condition because of the `require()` in L292.
2. in `DAOfiV1Pair.sol`, L321 and 328 are identical and can be moved before L319.
3. in `DAOfiV1Router01.sol`, in lines 121, 164, and 266 use `require()` instead of `assert()`.
4. in `IDAOfiV1Router01.sol`, commented out code in L61-72.

Test Results

Test Suite Results

```
DAOfiV1Factory
   ✓ createPair (291ms)
   ✓ createPair:gas (60ms)

DAOfiV1Pair: reverts
   ✓ initialize: (189ms)
   ✓ setPairOwner: (762ms)
   ✓ deposit: (960ms)
   ✓ withdraw: (1221ms)
   ✓ withdrawPlatformFees: (732ms)
   ✓ swap: initial requires (1048ms)
   ✓ swap: amount requires (1513ms)
   ✓ basePrice: (706ms)
   ✓ quotePrice: (725ms)
   ✓ getBaseOut: (726ms)
   ✓ getQuoteOut: (732ms)

DAOfiV1Pair: (y = x) m = 1, n = 1, fee = 0
   ✓ deposit: zero supply (340ms)
   ✓ deposit: 0 (717ms)
   ✓ deposit: 1 (715ms)
   ✓ withdraw: (1039ms)
   ✓ basePrice: (754ms)
   ✓ quotePrice: (716ms)
   ✓ getBaseOut: (717ms)
   ✓ getQuoteOut: (725ms)
   ✓ swap: quote for base and back to quote (1987ms)
   ✓ swap: verify price at supply (5661ms)
   ✓ withdrawPlatformFees: (1395ms)

DAOfiV1Pair: (y = 100x) m = 100, n = 1, fee = 0
   ✓ deposit: 0 (757ms)
   ✓ deposit: 1 (698ms)
   ✓ basePrice: (709ms)
   ✓ quotePrice: (700ms)
   ✓ getBaseOut: (697ms)
   ✓ getQuoteOut: (701ms)
   ✓ swap: quote for base and back to quote (2329ms)
   ✓ swap: verify price at supply (5711ms)
   ✓ withdrawPlatformFees: (1376ms)

DAOfiV1Pair: (y = 0.000001x) m = 0.000001, n = 1, fee = 0
   ✓ deposit: 0 (752ms)
   ✓ deposit: 1 (761ms)
   ✓ basePrice: (796ms)
   ✓ quotePrice: (749ms)
   ✓ getBaseOut: (771ms)
   ✓ getQuoteOut: (745ms)
   ✓ swap: quote for base and back to quote (1859ms)
   ✓ swap: verify price at supply (5516ms)
   ✓ withdrawPlatformFees: (1271ms)

DAOfiV1Pair: (y = x) m = 1, n = 1, fee = 3
   ✓ withdraw: including fees (1682ms)
   ✓ withdrawPlatformFees: (1381ms)


44 passing (1m)
```

```
DAOfiV1Router01: m = 1, n = 1, fee = 0
   ✓ addLiquidity: zero quote (90ms)
   ✓ addLiquidity: base and quote (117ms)
   ✓ addLiquidityETH: base and quote (118ms)
   ✓ removeLiquidity: (197ms)
   ✓ removeLiquidityETH: (196ms)
   ✓ basePrice: (844ms)
   ✓ quotePrice: (751ms)
   ✓ getBaseOut: (260ms)
   ✓ getQuoteOut: (742ms)
   ✓ swap: quote for base and back to quote (257ms)
   ✓ swap: multiple swaps (1634ms)
   ✓ swap: Ether for Tokens (237ms)
   ✓ swap: Tokens for Ether (248ms)


13 passing (41s)
```

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

```
ee14dac8f6f1464bd9432bfbd36679bb5841b92e1d5502e56fde3195eb9ab144 ./DAOfiV1Router01.sol
4afa750c582c586b4dfee2542feec2cb93de0775c18927f345ae691975937371 ./IERC20.sol
864c1ad51f8f8feb32bcad6f1b12e0dc7d1d5fa219184b5eaceb6a71de212138 ./IWETH10.sol
0c8cc56eaf4184223d987912277e36c00fc7cc62896ccef28836f58a63eb597b ./IERC2612.sol
98645a51686cc1a8be8c3392917a2240a5a2f8cd2a9e7062758ed1a178591d1c ./IDAOfiV1Router01.sol
f0a1fc1ccfdaaaae7673164da691b8284eaadc1f2dcf6a36545fecf11ba05d8f ./IWxDAI.sol
61f1d1100ac13e84d37e52f2c42c0589d728b4c920b4e3b4a9d4aa593b5352ac ./SafeMath.sol
0b051db1abf400cdf80209a67052bc6d3303510e1cd978be0bff0171fe1b7f51 ./DAOfiV1Library.sol
d1534e184fa3840b3b55b87d33e378fc54528b4b8fc3ffd5fbbc253d1d1f4599 ./WETH10.sol
66d5f9a588a6bde3339bf505937f59be6323dc2dfc85ef96fedd6aed8b2b2a96 ./ERC20.sol
e0c7450be19fa08e6b3e1f4d165a91839dee3ca634c370432b3911ba8f731527 ./WxDAI.sol
3effae0b6071a7a3c791ac01194134c0b4776c7edb85b79851ce48a32cc95d59 ./contracts/DAOfiV1Pair.sol
5718bdc0bcbdc41bd46eaabd7ca6dc117844fe4131f05c99e1f6065c11581458 ./contracts/DAOfiV1Factory.sol
4afa750c582c586b4dfee2542feec2cb93de0775c18927f345ae691975937371 ./contracts/interfaces/IERC20.sol
0e6f253993241262be7f1174a0e94a042285543ccc67a5fdb656f3fbc0a27430 ./contracts/interfaces/IDAOfiV1Pair.sol
4bbbaf4c3dcf38a8c76f4500cb9860825d2415811be2bdcc2184ac44391d3f99 ./contracts/interfaces/IDAOfiV1Factory.sol
61f1d1100ac13e84d37e52f2c42c0589d728b4c920b4e3b4a9d4aa593b5352ac ./contracts/libraries/SafeMath.sol
309ac3a0cd87107124b08f0157781be78016b1e3376afae8a49db4b1363236f8 ./contracts/test/ERC20.sol
```

Tests

```
831351edb2621ea620231cba838a67c15f86da6f0e0ed5f9ab93a1a3bf417176 ./DAOfiV1Pair.spec.ts
15901b3c439df950bff5a3a33a9187940f2060864892aa173e293903b2c0117c ./DAOfiV1Factory.spec.ts
718448b8a089a0d954fde2562d33b54b0b1a8dea2a63291172229f7ebbfb4dc3 ./fixtures.ts
e3351cf5fd566a36dec94b7d3c05ea625cde61c389dd49b7aee2c612e3626b84 ./utilities.ts
edf231422cd2a113e43211fcc434137d8cfe039cd2ed1a2bdf912002473c21dc ./test/DAOfiV1Router01.spec.ts
7608a15330bcd5d53e71dd1a8e603126e9fd0b313557b94524fc9604900afeaf ./test/shared/fixtures.ts
a212982eef68b3828a867dfea44f17499286d54e14d90d42968fd1a51553d8fd ./test/shared/utilities.ts
```

Changelog

- 2021-02-08 - Initial report
- 2021-02-24 - Revised report based on commits `daofi-v1-core/511bf39` and `daofi-v1-periphery/5976012`.

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.