Quantstamp Security Assessment Certificate

July 1st 2022 — Quantstamp Verified

Frabric DAO

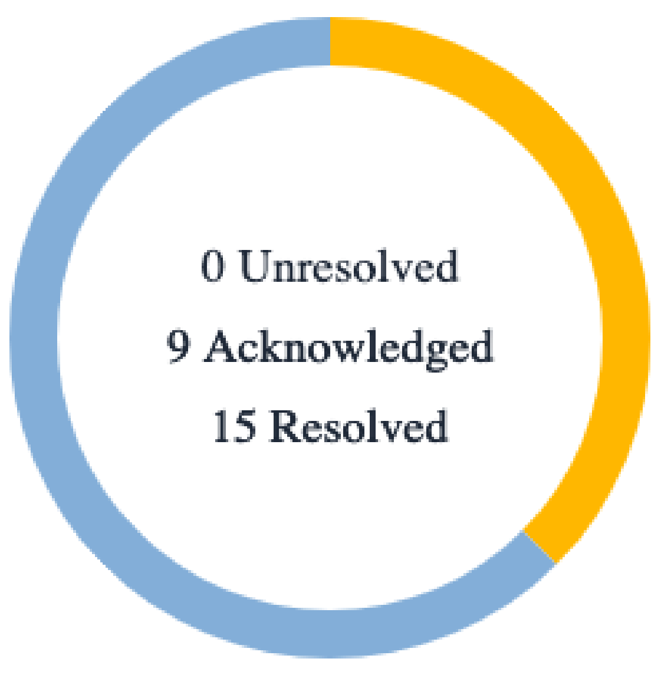
This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	DAO
Auditors	Andy Lin, Senior Auditing Engineer Fatemeh Heidari, Security Auditor Marius Guggenmos, Senior Research Engineer
Timeline	2022-06-09 through 2022-07-10
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	The Frabric Gitbook FrabricDAO contract Project
Documentation Quality	Medium
Test Quality	Medium
Source Code	

Repository	Commit
frabric-protocol (initial)	d78ebdee
frabric-protocol (re-audit)	fcff8a5

Total Issues	**24**	(15 Resolved)
High Risk Issues	**1**	(1 Resolved)
Medium Risk Issues	**2**	(2 Resolved)
Low Risk Issues	**8**	(5 Resolved)
Informational Risk Issues	**9**	(5 Resolved)
Undetermined Risk Issues	**4**	(2 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).
○ Fixed	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

Frabric protocol is a DAO protocol with several components, including `Frabric` (parent DAO), `Thread` (child DAO), `FrabricERC20` (governance token), `Auction`, `DEX`, and `Crowdfund` contracts. It includes all kinds of features of how participants can be whitelisted and interact with the DAOs and the governance tokens.
We found that the code has been developed with security in mind, as demonstrated by comments throughout the code base explaining design decisions and countermeasures against potential attacks. Due to the documentation lagging behind changes in the code, understanding the complex interactions of the contracts is fairly hard. Updated high-level documentation and NatSpec comments on functions and contracts would greatly help in that regard.
Re-audit update: The team has significantly improved the documentation and fixed or acknowledged all the reported issues.

ID	Description	Severity	Status
QSP-1	Lacking Specification	High	Mitigated
QSP-2	Miscalculation of the Super Majority	Medium	Fixed
QSP-3	Wrong Filled Amount on the Order	Medium	Fixed
QSP-4	Front-Run Buy Order	Low	Acknowledged
QSP-5	Thread Claws Back Tokens for Frabric	Low	Fixed
QSP-6	Risk of Colliding the Proposal ID with the Address Space	Informational	Fixed
QSP-7	DAO Proposals Cannot Be Completed After 2038	Low	Fixed
QSP-8	Anyone Can Choose Which Order to Cancel on Completed CancelOrder Proposal	Low	Acknowledged
QSP-9	Hardcoded Decimals in FrabricDAO	Low	Fixed
QSP-10	Crowdfund DoS Through Front-running	Low	Fixed
QSP-11	Global State Does Not Account for Removed Addresses	Low	Acknowledged
QSP-12	Missing Input Validation	Low	Mitigated
QSP-13	Proposing Non-Active Governor	Informational	Fixed
QSP-14	Implicit Token Action Intentions	Informational	Acknowledged
QSP-15	Not Matching Better Price with Limit Order DEX	Informational	Acknowledged
QSP-16	Force Withdrawal when Cleaning Up Orders for Removed Participants	Informational	Fixed
QSP-17	Lack of Post Deployment Check	Informational	Acknowledged
QSP-18	Missing Access Control on Burn Crowdfund Tokens	Informational	Fixed
QSP-19	`_setKYC` Function does not Check if the User Status is Removed	Informational	Acknowledged
QSP-20	Unlocked Pragma	Informational	Fixed
QSP-21	Preventing FrabricDAO's token action execution	? Undetermined	Fixed
QSP-22	Legacy Proposal Risk	? Undetermined	Fixed
QSP-23	Inconsistent Whitelist on Frabric Change	? Undetermined	Acknowledged
QSP-24	Partially Uninitialized Contract When Upgrading	? 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.

DISCLAIMER:
The team plans to deploy on L2s, such as Arbitrum. Please note that the current L2s are mostly still in beta phases. The L2 platform-specific issues are out of the scope of the current audit.

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 v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Since there are some issue on the Slither to run with this code base, we patched a few fixes locally:
   1. Follow this Github issue discussion to append `all_enums`.
   2. Expend the assertion of this line. Changed it to `assert isinstance(type_expression_found, (ElementaryTypeNameExpression, Identifier))`
3. Run Slither from the project directory: `slither .`

Findings

QSP-1 Lacking Specification

Severity: *High Risk*

Status: Mitigated

Description: The interaction patterns on the Frabric DAO are relatively complex, while at the same time, the documentation and specification are outdated. It is essential to have a precise specification of the features and responsibilities of each component and how they could interact with each other so the audit team can verify whether the implementation is following the spec.

Recommendation: From our discussion, we know that the team has been working on updating the documentation. We recommend providing a clear spec on each DAO actor and their potential interactions. Also, the team should consider a document-first approach when developing new features, so there will always be a clear spec and documentation to work on when new features are added.

Update: The team has updated the non-technical documents in https://the-frabric.gitbook.io/thefrabric/ and added Natspec to the smart contracts. The team also states that the documentation is still working in progress but will continue to improve.

QSP-2 Miscalculation of the Super Majority

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `dao/DAO.sol`

Description: In `DAO.sol:queueProposal` and `Dao.sol:cancelProposal` functions, when calculating the supermajority of votes, it assumes that 50% of votes will offset to 0, so an extra 16% will lead to a supermajority of 66%. However, the math is wrong here. When the votes end up to be exact 16% positive, there are `(100 - 16) / 2 = 42` percent of total votes on the negative side and `(100 - 16) / 2 + 16 = 58` on the positive side. The positive votes are just 58% of the total votes instead of the expected 66%.

Exploit Scenario: Alice votes with 58 votes on "Yes," and Bob votes with 42 votes on "No". The DAO will queue the proposal despite not passing the supermajority (66%).

Recommendation: Use 33% instead of 16%. With 33%, it will need `(100 - 33) / 2 + 33 = 66` percent of positive votes.

- In `DAO.sol:L262`, instead of `passingVotes = int112(proposal.totalVotes / 6)`, it should be `passingVotes = int112(proposal.totalVotes / 3);`.
- In `DAO.sol:L326`, instead of `passingVotes = int112(proposal.totalVotes / 6)`, it should be `passingVotes = int112(proposal.totalVotes / 3);`.

Update: The team fixed as recommended in the commit `e22019b6b7`

QSP-3 Wrong Filled Amount on the Order

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `erc20/IntegratedLimitOrderDEX.sol`

Description: In `IntegratedLimitOrderDEX.sol:_fill`, it directly returns 0 when `h==0` on `L128-132` during the removal of the non-whitelisted traders' orders. However, the new order can match other orders before those non-whitelisted traders' orders. The implementation will update the token and the `tradeTokenBalances` and `locked` mappings accordingly on `L149-156`. Returning 0 here will also cause `IntegratedLimitOrderDEX.sol:_action` to place the new order with the wrong amount.

Exploit Scenario: The same scenario is valid for buy orders

1. User `A` creates a buy order with price `P` and amount `10` and is placed at index 0 of `_points[P].orders[0]`
2. User `B` creates a buy order with price `P` and amount `5` and is placed at index 0 of `_points[P].orders[1]`
3. User `A` is removed and is not whitelisted anymore.
4. User `C` calls the `sell` function with price `P` and amount `20`
5. Since there are buy orders at the same price, the `_fill` function is called, and in the first iteration of the `for` loop, part of the order is filled with `_points[P].orders[1]`. User `B` and User `C` will receive their tokens.
6. In the next iteration, since User `A` is not `whitelisted` anymore, the execution falls into the `while` loop, and since the condition `h == 0` is met, the `_fill` function returns `0` instead of `5` as expected.
7. Consequently, a sell order from User `C` with the amount `20` is placed, allowing them to receive the original order amount of `20` when new `buy` orders are placed despite receiving `5` tokens already.

Recommendation: Return the `filled` variable instead of `0`.

Update: The team fixed as recommended in the commit `f54a8f048`

QSP-4 Front-Run Buy Order

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `erc20/IntegratedLimitOrderDEX.sol`, `erc20/DexRouter.sol`

Description: The function `IntegratedLimitOrderDEX.sol:buy` is designed so that the user should transfer the funds to the contract before calling this. Inside the `buy` function, it checks the difference between the current balance of the `IntegratedLimitOrderDEX` contract and the last recorded `tradeTokenBalance`. However, an attacker can front-run by placing the attacker's own `buy` transaction between the user's fund-transfer transaction and the `buy` transaction. The implementation will treat the user's funds as the attacker's buying money.

Exploit Scenario:

1. Alice sends a transaction to transfer 100 tokens to the `IntegratedLimitOrderDEX` contract.
2. Bob immediately places a buy order. Bob can use those 100 tokens on his order.

Recommendation: We know the implementation expects `DEXRouter.sol` to proxy the `buy` call. Using `DEXRouter.sol:buy` ensures the atomicity of the funds transfer and the buy order creation. However, `IntegratedLimitOrderDEX.sol:buy` is an open public function. We recommend restricting only the `DEXRouter.sol` can call the `buy` function or having embedded `safeTransferFrom` inside the `IntegratedLimitOrderDEX.sol:buy` function to retrieve the user's fund. In the case of having embedded `safeTransferFrom` call, the `DEXRouter.sol` will need to approve the `IntegratedLimitOrderDEX.sol`.

Update: The team prefers to keep the same code as it is more gas-efficient, and Uniswap uses the same pattern. As Uniswap uses a similar pattern (code), and there is a router contract already, we decided to lower the severity to low. However, we recommend, in this case, adding a warning to the Natspec of the `IntegratedLimitOrderDEX:buy` function.

QSP-5 Thread Claws Back Tokens for Frabric

Severity: *Low Risk*

Status: Fixed

File(s) affected: `thread/Thread.sol`, `thread/ThreadDeployer.sol`, `erc20/Timelock.sol`

Description: The code documents in `erc20/Timelock.sol:L17-19` and `thread/Thread.sol:L274-283` state that there should be sufficient protection around the upgraded `Thread` to claw back the thread tokens in the `Timelock` for the `Frabric`. The protection is to grant enough time for the `Frabric` to react on and try to sell the thread tokens before `Thread` gets upgraded. The time granted is in `Thread.sol:L284` with the formula `upgradesEnabled = block.timestamp + IDAOCore(frabric).votingPeriod() + (1 weeks);`. However, the `frabric` here is the new one being set in `Thread.sol:L267` with `_setFrabric(_frabrics[id])`. As a result, the thread can provide a new `Frabric` with the `votingPeriod` being set to 0 to reduce the actionable time for the original `Frabric`.

Exploit Scenario:

1. `t0`: Alice proposes an ecosystem leave to the `Thread` with a new `Frabic` with a voting period set to 0.
2. `t1 = t0 + 1 week (thread voting period) + 48 hours (queue period)`: The proposal is passed and completed. Now, `upgradesEnabled` is a week.
3. `t1`: After Alice completes the thread proposal, Bob tries to save the thread tokens in the `Timelock` contract and calls `ThreadDeployer.sol:claimTimelock` to move tokens from the `Timelock` to the `Frabric`.
4. `t1`: Bob proposes a token action proposal and waits for 2 weeks to vote (`Frabric` voting period is 2 weeks) plus 48 hours in the queue.
5. `t1 + 1 week (upgradesEnabled)`: Alice proposes upgrading the thread token to expose functions to claw back the tokens.
6. `t2 = t1 + 2 weeks (frabric voting period) + 48 hours (queue period)`: Bob executes the token action proposal, and he sells those tokens on DEX
7. `t2`: at the same time, Alice completes her proposal. Now the thread token code has been upgraded.
8. `t2`: Alice can do anything on the token despite listing in the DEX. There is no time for people to buy the tokens from the DEX.

Recommendation: Run this line `upgradesEnabled = block.timestamp + IDAOCore(frabric).votingPeriod() + (1 weeks);` before setting the `frabric` to a new one in `Thread.sol:_completeSpecificProposal`.

Update: The team fixed by setting the `frabricVotingPeriod` in the `initialize` function instead of getting from `IDAOCore(frabric).votingPeriod()` on demand to avoid new frabric reducing the voting period. Following is the team's statement:

```
The provided recommendation was not followed so Threads may organize as they choose without being forced to enable upgrades, which would be a step towards potentially endangering their users.
```

QSP-6 Risk of Colliding the Proposal ID with the Address Space

Severity: *Informational*

Status: Fixed

File(s) affected: `frabric/Frabric.sol`

Description: `Frabric.sol:approve` has the input `approving` with the type of `address`. However, in `Frabric.sol:L355-360`, the `approving` is not an address but a proposal id. It uses the value under `approving` as the `id` to fetch the participant data from the proposal. Despite the low chances of this, there is the risk of the proposal id eventually colliding with an active address space. The proposal id will continue to grow naturally and increase the risk of colliding with an active participant address.

Recommendation: We recommend having another input: `proposal_id`. The `approve` function should verify only one of the `approving` or the `proposal_id` is supplied (non zero). We then use the `proposal_id` to fetch data from the `passed` and `_participants` mappings.

Update: The team states that because there is the extra parameter `ParticipantType`, the space collision is not a concern. Quantstamp decides to lower the severity to "information". The team also added zero address validation to the `approve` function. Still, we recommend adding a code document to clarify that although the implementation declares the `approving` variable as the `address` type, it will be used as proposal ID (uint256) if the `pType` variable is NULL.

QSP-7 DAO Proposals Cannot Be Completed After 2038

Severity: *Low Risk*

Status: Fixed

File(s) affected: `dao/DAO.sol`

Description: The function `queueProposal` stores the start time in `proposal.stateStartTime` as `uint32(block.timestamp)` (L276), which will be truncated in the year 2038 and start from 0. This timestamp is later used in `completeProposal` to check whether the proposal can be completed and will always result in an error being thrown.

```
uint256 end = proposal.stateStartTime + queuePeriod;
if (block.timestamp < end) {
    revert StillQueued(id, block.timestamp, end);
}
```

Recommendation: Since the `proposal.stateStartTime` struct member is already a `uint64`, perform the assignment with `block.timestamp` cast to `uint64` instead of `uint32`.

Update: The team fixed it as recommended in the commit `55fd04e1b70`.

QSP-8 Anyone Can Choose Which Order to Cancel on Completed CancelOrder Proposal

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `dao/FrabricDAO.sol`

Description: The `DAO.completeProposal(uint256 id, bytes calldata data)` function can be called by anyone with an arbitrary `data` argument. The only place where `data` is used is in `FrabricDAO._completeProposal`, where it is decoded to an index that will be used as an argument to the `cancelOrder` function (L285). This means the person to call `completeProposal` can decide which order to cancel.

Recommendation: A naive solution is to lock down the index to cancel on the proposal stage. One issue with this solution is that the index of the order array can be changed if any order deletion occurs between the order cancel proposal stage and the completion stage. To solve this entirely, the team should consider using a `mapping` or an `EnumerableMap` inside `PricePoint.orders` instead of an array.

Update: The proposal locks down the cancelation to a specific price point even though the caller can give the order ID arbitrarily. Also, using EnumerableSet will incur an extra gas cost. Following is the full statement from the team:

```
This was handled by the ILO DEX returning true if your order was canceled, yet false if an arbitrary order was canceled. While this does let the executor specify which order to cancel for a given price point, they must
cancel an order of the DAO successfully. If there are multiple orders, the DAO is expected to place multiple cancellations. If the DAO wants to keep some of its tokens at that price, it is expected to have created them 1-
wei off at the start OR re-list their tokens.
```

```
An iterative approach was taken for effective trade matching, yet an iterative approach isn't feasible for cancellation due to the array length potentially causing excessive gas costs (whereas if a price point is spammed,
anyone can clear orders from it or users can move to a price point distinct by 1 wei). An EnumerableSet would always copy the entire set of keys into memory, potentially not allowing any trades to go through however, while
significantly increasing gas costs.
```

QSP-9 Hardcoded Decimals in FabricDAO

Severity: Low Risk

Status: Fixed

File(s) affected: `dao/FabricDAO.sol`

Description: `FabricDAO.proposeTokenAction` accepts an arbitrary `token` address but hardcodes the token's decimals as `1e18`. This could lead to tokens that cannot be sold via a DAO proposal if the token's decimals are significantly different from `1e18`.

Recommendation: Query the token's decimals instead of using a hardcoded value.

Update: The team fixed it as recommended in the commit `f718ab128`.

QSP-10 Crowdfund DoS Through Front-running

Severity: Low Risk

Status: Fixed

File(s) affected: `thread/Crowdfund.sol`

Description: For a `Crowdfund` to be finalized, it must be fully funded before the governor can call the `execute` function. During the funding period, anyone can withdraw their funds, and only up to the funding target can be provided. To stop the crowdfund from being executed, an attacker can front-run the governor's `execute` call and withdraw their funds, making the transaction revert.

Exploit Scenario:

1. An attacker deposits any amount into a crowdfund contract.
2. The attacker waits until the contract is fully funded.
3. The attacker monitors the mempool for transactions made by the governor and front-runs any calls to `execute` with a call of `withdraw`.
4. The `execute` will fail and the attacker can deposit their amount immediatly after. Repeat steps 3 and 4.

Recommendation: When the target amount has been reached by a call to `deposit`, consider disabling `withdraw` for a short amount of time to make sure a malicious actor cannot DoS the finalization.

Update: The team fixed it as recommended in the commit `fd8656e`.

QSP-11 Global State Does Not Account for Removed Addresses

Severity: Low Risk

Status: Acknowledged

Description: In the `FabricWhitelist` contract, the `whitelisted(address person)` function checks if the `person` is whitelisted or the `global` is true. In all the situations, the function returns `false` for a person with a removed state even if the global state is `true`.
Also, note that in the current implementation, only the test contract will be able to set the `global` to true. The `_setGlobal` function is never being used.

Recommendation: First, please consider removing the unused variable `global` and the `_setGlobal` function. Future upgrades can add those back when needed. If they are to keep, please clarify if the implementation is intentional, or revise the conditions always to return `true` when `global` is `true`.

Update: The team acknowledged that they intend to leave the code here to avoid future lengthier discussions. Also, the global acceptance should void the whitelist but not the participant removal. So the code should not just return true when the `global` variable is true.

QSP-12 Missing Input Validation

Severity: Low Risk

Status: Mitigated

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error.

1. In the following functions, arguments of type `address` may be initialized with value `0x0`:
    - [ack] `DAO.__DAO_init(...)` ln#60,
    - [fixed] `FabricERC20.initialize(...)` ln#29,
    - [ack] `Bond.initialize(...)`
    - [ack] `Crowdfund.initialize(...)`
    - [ack] `Thread.initialize(...)`
    - [fixed] `ThreadDeployer.initialize`
2. [fixed] In the `DAO.vote(...)` function (`L204`), check the length of arrays `ids` and `votes` to be equalå

Recommendation: We recommend adding the relevant checks.

Update: Despite the team not adding validation to all functions mentioned directly, for most of the functions without changes, the nature of the inheritance calls mitigates those. For instance, the team adds the non-zero address validation in the `ThreadDeployer`, and it will ensure passing validated addresses to other downstream `init` functions without needing extra validations.

QSP-13 Proposing Non-Active Governor

Severity: Informational

Status: Fixed

File(s) affected: `thread/Thread.sol`

Description: The function `Thread.sol:proposeFrabricChange` does not check with the modifier `Thread.sol:viableGovernor` on the `governor` input. Without the modifier, one can propose a non-active governor of the new frabric.

Recommendation: Add the modifier `Thread.sol:viableGovernor` to the `Thread.sol:proposeFrabricChange` function.

Update: The team fixed it as recommended in the commit `599ece39b46`.

QSP-14 Implicit Token Action Intentions

Severity: Informational

Status: Acknowledged

File(s) affected: `dao/FrabricDAO.sol`

Description: It is non-intuitive to understand from the parameters of the `FrabricDAO.sol:proposeTokenAction` inputs to know which specific token action the proposal is attempting to do. If we look at `FrabricDAO.sol:L280-334`, the logic inside `FrabricDAO.sol:_completeProposal` for the token actions is complex and involves multiple token activities: mint, cancel an order on DEX, transfer, sell on DEX, and list to the auction. Deciding which specific action to perform implicitly depends on `target`, `price`, and `amount`. The logic is hard to follow and prone to errors on any potential code upgrades.

Recommendation: We recommend adding an `Enum` to specify the token action type and refactor the function `FrabricDAO.sol:proposeTokenAction` and `FrabricDAO.sol:_completeProposal` to use the "token action type" to determine how to validate the input and how to perform the action.

Update: The team decides to keep the code due to the code size limit. However, the team has improved the Natspec documentation for the function.

QSP-15 Not Matching Better Price with Limit Order DEX

Severity: Informational

Status: Acknowledged

File(s) affected: `erc20/IntegratedLimitOrderDEX.sol`

Description: Usually, a user can execute a limit order on the exact price or a better order price. For instance, Investopedia defines a limit order as "A limit order is a type of order to purchase or sell a security at a specified price or better" (link). `IntegratedLimitOrderDEX` only allows matching the exact price but not with a better price. The exact-match requirement can confuse the user.

Recommendation: If this restriction is business-wise accepted, document this difference to the user.

Update: Due to EVM constraints, it is hard to implement limit orders with better price matching. The team has added documentation in the `IntegratedLimitOrderDEX` contract and will note this to the user.

QSP-16 Force Withdrawal when Cleaning Up Orders for Removed Participants

Severity: Informational

Status: Fixed

File(s) affected: `erc20/IntegratedLimitOrderDEX.sol`

Description: The `IntegratedLimitOrderDEX.sol:cancelOrder` will automatically withdraw the funds of the user with `_withdrawTradeToken(msg.sender);`. However, users can not only cancel their orders but also cancel orders with an already-removed trader. The call of `_withdrawTradeToken(msg.sender)` will run no matter whether it cleans the user's order. This forces whoever wants to help clean up the orders to withdraw.

Recommendation: Consider removing the withdrawal feature on canceling the order if that is not necessary. If the automatic withdrawal is preferred, consider withdrawing only when the `ours` variable is not true.

Update: The team fixed it as recommended in the commit `a3375022e7`.

QSP-17 Lack of Post Deployment Check

Severity: Informational

Status: Acknowledged

File(s) affected: `scripts/*.js`

Description: The whole Frabric protocol is composed of multiple contracts, and many of those have dependencies on others. The contract setup for the entire system is non-trivial. A simple mistake in the deployment script can cause a significant impact on the misconfiguration.

Recommendation: Given the complexity of the contracts, we recommend listing down the spec for each contract's dependencies and adding a verification script after all deployments according to the spec.

Update: The team agrees on the importance and will work on it after the audit.

QSP-18 Missing Access Control on Burn Crowdfund Tokens

Severity: Informational

Status: Fixed

File(s) affected: `thread/Crowdfund.sol`

Description: In the `Crowdfund.burn(address depositor)` function, there is no control on the `msg.sender` being the same address as `depositor`. Anybody will be able to call this function for any address. Luckily, this is actually not just burning the crowdfunding token but also transferring the thread token as the crowdfunding result.

Recommendation: Please consider renaming the function to `redeem` instead of `burn`. Also, we recommend only allowing the depositor itself (`msg.sender`) or the `governor` to be able to redeem for the specific `depositor`.

Update: The `burn` function is renamed to `redeem`. However, it is by design to allow anyone to call. Here is the explanation from the team:

```
Anyone is allowed to call redeem, by design, in order to allow anyone, such as organizations like Fractional Finance, to force a migration, closing out the Crowdfund and its useless tokens for the active Thread whose tokens
have utility.
```

QSP-19 `_setKYC` Function does not Check if the User Status is Removed

Severity: Informational

Status: Acknowledged

File(s) affected: `erc20/FrabricWhitelist.sol`

Description: In `FrabricERC20` contract, the `_setKYC` function checks to only set whitelisted users' status to `Status.KYC`, but it does not revert if the user state is `Removed`, and executes the `kyc[person] = hash;` statement.

Recommendation: Ensure that the user is not removed before setting a `KYC` hash for a person.

Update: They need this for the KYC organization to update information. Following is the statement from the team:

```
This is to allow KYC organization to update information, on-chain, in a verifiable way, as situations develop with malicious actors. Given the immutable history of the chain, and the non-active participation of those
removed, this enables another communication channel while not reducing security.
```

QSP-20 Unlocked Pragma

Severity: Informational

Status: Fixed

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. 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".
Also, there are multiple declaration on the pragma version within different contracts: `['>=0.8.9', '^0.8.9']`.

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

Update: Aside from the interfaces, all contracts have locked the version to `0.8.9`.

QSP-21 Preventing FrabricDAO's token action execution

Severity: Undetermined

Status: Fixed

File(s) affected: `dao/FrabricDAO.sol`, `erc20/FrabricERC20.sol`, `frabric/Frabric.sol`, `thread/Thread.sol`

Description: The `FrabricDAO.sol:proposeParticipantRemoval` allows any participant to freeze the DAO token for another participant for more than 10% of the votes. One can use this to freeze the `Frabric` contract. Once the `Frabric` contract is frozen, it cannot execute the token action proposals as part of the `FrabricDAO.sol:_completeProposal` function. The reason is that the logic in `FrabricERC20.sol:_beforeTokenTransfer` will prevent the transfer from a frozen address.
The same applies to the `Thread` contract because `Thread` also inherits `FrabricDAO`. Despite `Thread` having extra validation with `irremovable` in `Thread.sol:proposeParticipantRemoval`, the `Thread` contract itself is not part of the `irremovable`.

Recommendation: The implementation should not freeze the DAO contract itself.

Update: It restricts propose participant removal of the `address(this)` in the commit `88c19687c`.

QSP-22 Legacy Proposal Risk

Severity: Undetermined

Status: Fixed

File(s) affected: `dao/DAO.sol`, `dao/FrabricDAO.sol`, `frabric/Frabic.sol`, `thread/Thread.sol`

Description: Once the community has voted on the proposal, it can potentially be left non-executed for a long time unless the original proposer decides to withdraw the proposal. Eventually, it can conflict with another new proposal and cause an unexpected impact. Also, there are some proposals that only a specific actor can complete (e.g., Governor change and Dissolution of the thread). If the privileged actor decides not to execute the proposal, it can be a blocking factor for the DAO to continue running. Last but not least, if the DAO community realizes some issue with the proposal after the voting period and the original creator has lost the key, the community cannot cooperate in canceling the proposal unless going with a risky operation such as a code upgrade.

Exploit Scenario:

1. The thread community votes and passes on a `EcosystemLeaveWithUpgrades` proposal.
2. Somehow, the new governor refused to complete the proposal.
3. Now, the thread is left in an awkward state and cannot upgrade itself (it need the `EcosystemLeaveWithUpgrades` completion to upgrade). The only way out seems to be the original Frabric help adding a specific code to the release channel.

Recommendation: The proposal should have a deadline for execution. The user can only complete the proposal before the deadline. Also, once the deadline passes, everyone can cancel the proposal.

Update: The team fixed by adding a `lapsePeriod` that allows executing the proposal within 48 hours after queued in the commit `340d0771`.

QSP-23 Inconsistent Whitelist on Frabric Change

Severity: Undetermined

Status: Acknowledged

File(s) affected: `erc20/IntegratedLimitOrderDEX.sol`, `erc20/Auction.sol`, `erc20/FrabricWhitelist.sol`, `erc20/FrabricERC20.sol`, `thread/Thread.sol`

Description: For a thread token, it will automatically inherit the whitelist of the parent token (the Frabric token) in the code of `FrabricWhitelist.sol:whitelisted`. However, when the thread changes its Frabric, the implementation will change the "parent" token in `Thread.sol:_setFrabric`. The "parent" token change causes the whitelisted participant on the old Frabric to become un-whitelisted to the thread. Now, anyone can call `FrabricERC20.sol:triggerRemoval` to remove that participant. Also, if there are remaining bids on the `IntegratedLimitOrderDEX`, those can cause the participant to be removed in `IntegratedLimitOrderDEX.sol:_fill` or `IntegratedLimitOrderDEX.sol:cancelOrder`. To highlight the impact, the attacker can remove the old `Frabic` and the `Auction` contracts from the thread after a Frabric change. Removing the old `Frabric` will bypass the `Thread.sol:irremovable` list that contains the original `Frabric`. It prohibits the `Frabric` from performing token actions to safeguard the thread tokens in the `Timelock` contract. Also, removing the `Auction` will cause all the bidders to fail to withdraw from the auction as transferring from the `Auction` contract will be prohibited by `FrabricERC20.sol:_beforeTokenTransfer`. Note that currently, there is no way to update the auction for the thread, so the thread token will always be using the original auction.

Exploit Scenario:

1. The thread community passes and completes the Frabric change proposal.
2. An attacker of the thread community calls `Thread.sol:triggerRemoval` to remove the old `Frabric` and `Auction`.
3. The attacker proposes a `EcosystemLeaveWithUpgrades`. It passes and completes.
4. Now, the thread community upgrades the thread token code and get those thread token back with the upgraded code.

Recommendation: We recommend mitigating this with some potential directions:

1. Confirm whether a business justification exists for having a `FrabricChange` proposal. From the Fractional finance team discussion, it seems ambiguous on the use cases for this. Suppose the thread can use other proposal types such as `EcosystemLeaveWithUpgrades` or `Dissolution` for a similar use case to run their DAO independently with the Frabric from Fractional finance. In that case, the team should consider removing the feature of the `FrabricChange` proposal. Note that this assumes the thread will run `EcosystemLeaveWithUpgrades` carefully to upgrade to a code that removes the dependency on the `Frabric` and the `Auction` contracts.
2. If there is a business justification for having a `FrabricChange` proposal, we recommend doing the following:
   1. The Threads should not depend on the Frabric's whitelist and maintain their own, so on a `FrabricChange` event, the thread's whitelist won't become empty. However, if the team prefers to keep the whitelist dependency, please provide the guidances/documents or some scripts/SDKs to verify whether or not the new Frabric has a competent whitelist. So the Thread community can judge whether the new Frabric will cause unexpected participant removal after the change or not.
   2. Update the `auction` used in the new `Frabric` as part of the frabric changes.

Update: The nature of the supermajority vote required by the frabric change proposal mitigates the risk. Following is the statement from the team:

```
FrabricChange is an intended offering, requiring a supermajority due to its effects. The Auction contract remains the same to not directly change code of the system when a change occurs, solely the data in the system.
```

QSP-24 Partially Uninitialized Contract When Upgrading

Severity: Undetermined

Status: Acknowledged

File(s) affected: `beacon/Beacon.sol`

Description: The upgrade process is a 2-step process. In the first transaction, the implementation address is updated. In a second transaction, `triggerUpgrade` must be called for each proxy that is using the `Beacon` contract, which will call the `upgrade` function on the proxy with the calldata that was specified in the first upgrade step. During steps 1 and 2, the contract will be using the updated code while new functionality has potentially not been initialized yet.

Recommendation: Consider implementing a mechanism that requires `triggerUpgrade` to be called before the beacon will resolve the address to the new contract (`implementation` could either throw an exception or return the old address). This could be opt-in so only upgrades that have to be initialized before being usable have this requirement.

Update: The team clarified that the Frabric contract, the only contract with this two-phase upgrade process, has guards to ensure new functionality was only used once upgraded, which is the intended flow. The team also states that the change will significantly burden Beacons, potentially making them no longer EIP-165 compliant.

Automated Analyses

Slither

Slither analyzed all contracts with 78 detectors, and 308 results were found. The majority of the issues alerted by Slither are false positives. We have embedded the valid ones into our issues or best practices sections.

Code Documentation

- Add the comment/documentation on the expected owner of the contracts:
  - [fixed: L49] `ThreadDeployer`: the owner will be the `Frabric` contract.
  - [fixed: L30] `Bond`: the owner will be the `Frabric` contract.
  - [fixed: L37] `FrabricERC20`: the owner will be the `Frabric` or the `Thread` contract.
- [fixed] Provide documents on the expected future upgrade steps. It should include the design of the "release channel" concept and its sample usages. **Update:** the team has added documentation in the non-tech doc.
- Add a code document for `frabic/Frabic.sol:approve` to indicate that we should call this function after executing the participant proposal.
- [ack] Remove or update the comment regarding struct optimization in `erc20/Auction.sol:L31-35`. It states `// uint96 amounts would be perfectly packed ...` while none of the fields in the `AuctionStruct` is using `uint96`. **Update:** the team responded that the word "would" here emphasizes that it could but did not

Adherence to Best Practices

- Consider writing tests in typescript with automatically generated types from Solidity using typechain.
- [fixed] Move the computation of `tenPercent` in `DAO.cancelProposal` (L307) outside of the loop to avoid recomputing it on each iteration.
- [fixed] In `dao/DAO.sol:L136-138`: consider reusing `_voteUnsafe(uint256 id, address voter)` instead inside the `_createProposal` function.
- [fixed] Consider removing `normalizeRaiseToThread(1);` in `Crowdfund.sol:initialize`. The `ThreadDeployer` will call it right after initialization in `ThreadDeployer.sol:L190`, and make this check unnecessary. Also, the function `normalizeRaiseToThread` will never overflow when given `1` as input. **Update:** instead of checking `normalizeRaiseToThread(1)`, it runs `normalizeRaiseToThread(target)` and reuse the returned value in ThreadDeployer
- [ack] In `thread/Thread.sol:L157`, consider renaming the revert error from `ProposingUpgrade` to something like `InvalidUpgradeProposal` to indicate the failure. **Update:** the team states that this should be clear enough as it is reverting.
- [ack] In `dao/DAO.sol:vote`, the implementation should be consistent on handling the case of `actualVotes == 0` (`L213`) and `votesI == 0`(`L225`). They should both call `DAO.sol:_voteUnsafe` to trigger a `Vote` event with `VoteDirection.Abstain` or both do nothing. **Update:** The team clarified that these are two distinct cases distinguished by presence v usage.
- [fixed] In `fabric/Bond.sol:_beforeTokenTransfer`, the implementation could run the validation of `L56-58` first before calling `super._beforeTokenTransfer(from, to, amount);` to save some gas when it fails.
- [ack] Provide a `Null` value as the first declaration of `Enum`. By default, Solidity will initiate a variable with an empty value. For `Enum`, it will be the first value of the declaration. Without having an explicit `Null` value on the `Enum`, this could lead to undesired default status. **Update:** the team states the following: "VoteDirection.Abstain is intended as equivalent to Null. Crowdfund's launch as Active, so they cannot have a Null state. ProposalTypes could have a Null, yet do not benefit from it given the checks on ProposalState, which does have a Null."
  - `IDAO.sol:VoteDirection`
  - `IFrabricDAO.sol:CommonProposalType`
  - `IInitialFrabric.sol:FrabricProposalType`
  - `ICrowdfund.sol:State`
  - `IThread.sol:ThreadProposalType`
- [ack] Replace magic numbers with a constant variable. **Update:** The magic values have been further documented yet not converted to constants at this time.
  - `thread/ThreadDeployer.sol:L68`: provide a constant for the magic number zero in `variant == 0`.
  - `thread/ThreadDeployer.sol:L163`: provide a constant for the magic number zero in `variant != 0`.
  - `dao/DAO.sol:L73`: provide a constant for the magic number `10` in `DAO.sol:requiredParticipation` function.
  - `dao/DAO.sol:L157`: provide a constant for the magic number `10` in `DAO.sol:_voteUnsafe` function.
  - `dao/FrabricDAO.sol:L76`: provide a constant for the magic number `100` in `FrabricDAO.sol:__FrabricDAO_init`. Ideally, specify this is for percentage, so the maximum is 100 (percent).
  - `frabric/Frabric.sol:L338`: provide a constant for the magic number `6` in `Frabric.sol:vouch`. Also, provide some background on how the team chose 6.
- Tighten the validations
  - [fixed] `thread/ThreadDeployer.sol:validate`: check that `data.length` should be exact 64 and verify the decoded token address and the target amount is non zero.
  - [ack] `thread/Thread.sol:proposeUpgrade`: check that the `instance` should be `address(this)` so that it only proposes to upgrade itself as the specific instance. The upgrade to another `instance` should fail when calling `Beacon.sol:upgrade` when as not authorized eventually. **Update:** Thread proposeUpgrade should not have its instance locked. It can theoretically own other contracts and already does own one other contract (their ERC20).
  - `frabric/Frabric.sol:validateUpgrade`: check the `data.length` to be 96, and the third decode address (`kyc` address) should not be zero.
- [ack] Consider explicitly stating the inherited contract name when using functions or variables from the parent contract for better readability and traceability: **Update:** The team prefers to keep the style.
  - `thread/Crowdfund.sol:L26`: declare as `ICrowdfund.State public state;`.
  - `thread/Crowdfund.sol:L196`: `DistributionERC20._distribute` instead of `_distribute`.
- [fixed] Consider removing `freezeUntilNonce` from the following functions: `FrabricDAO.sol:proposeParticipantRemoval`, `Thread.sol:proposeParticipantRemoval`, and `IFrabic.sol:proposeParticipantRemoval`. Inside the implementation, it can use the `expectedNonce` variable in `FrabicDAO.sol:L242` instead.
- [fixed] In `erc20/Auction.sol:L104`, the `_nextID++` would incur a gas cost of updating the storage on every looping step. The implementation can update the storage just once after the for loop and use the original `_nextID` value plus the `i` number of the for loop to derive the `id` for the auction array.
- [fixed] Lower the visibility of `beacon/Beacon.sol:triggerUpgrade` from `public` to `external`.
- [fixed] Consider adding an event in `erc20/Auction.sol:complete`. In `L227`, there should be an event when the try-catch of the `burn` fails.

Test Results

Test Suite Results

```
Compiled 82 Solidity files successfully


  Beacon
    ✓ should have the right name and version
    ✓ should have the right beacon name and number of release channels
    ✓ should allow upgrading release channel 0 (61ms)
    ✓ should resolve an address via release channel 0
    ✓ should let your set you own code (48ms)
    ✓ should let you set code of contracts you own (66ms)
    ✓ shouldn't let you set code of contracts you don't own, even as the owner (114ms)
    ✓ should support upgrading with data (125ms)
    ✓ should support triggering upgrades (65ms)
    ✓ shouldn't support triggering upgrades for the wrong version (49ms)
    ✓ should hit full code coverage on TestUpgradeable (65ms)

  SingleBeacon
    ✓ should have the right amount of release channels
    ✓ should allow upgrading release channel 0
    ✓ should only allow upgrading release channel 0

  Composable
    ✓ should have the correct name
```

```
      ✓ should have the right version (105ms)
      ✓ should be EIP-165 compliant
      ✓ should support its own interface

    DAO
      ✓ should have initialized correctly
      ✓ should have 10% required participation
      ✓ shouldn't expect participation from tokens it holds (57ms)
      ✓ should check canPropose
      ✓ should create proposals and automatically vote (154ms)
      ✓ should create proposals requiring a supermajority (50ms)
      ✓ should let you partially vote (52ms)
      ✓ should let you vote with more than you have yet correct it (120ms)
      ✓ shouldn't let anyone withdraw proposals
      ✓ should let the proposal creator withdraw an active proposal
      ✓ shouldn't let you queue proposals still being voted on
      ✓ shouldn't let you queue net positive proposals which require a supermajority (47ms)
      ✓ should let you queue passing proposals
      ✓ should let you queue passing supermajority proposals
      ✓ should let you withdraw queued proposals
      ✓ shouldn't let you cancel proposals which have enough votes
      ✓ shouldn't let you cancel proposals with voters who voted no
      ✓ shouldn't let you cancel proposals with repeated voters
      ✓ should let you cancel proposals which no longer have enough votes (67ms)
      ✓ shouldn't let you complete proposals which are still queued
      ✓ should let you complete proposals
      ✓ should delete the proposal, which leaves behind the vote map
      ✓ shouldn't let you withdraw completed proposals
      ✓ shouldn't queue proposals with insufficient participation (108ms)

    FrabricDAO
      ✓ should have initialized correctly
      ✓ should have paper proposals (207ms)
      ✓ should allow upgrading
      ✓ should allow transferring tokens (135ms)
      ✓ should allow selling tokens on their DEX (139ms)
      ✓ should allow cancelling orders on a DEX (128ms)
      ✓ should allow selling tokens at auction (175ms)
      ✓ should allow minting tokens (178ms)
      ✓ should allow minting and selling on the DEX (167ms)
      ✓ should allow minting and selling at Auction (207ms)
      ✓ should allow removing participants (951ms)

    Auction
      ✓ should let you create auctions (132ms)
      ✓ should let you create auctions in the future (128ms)
      ✓ should let you create auctions in batches (463ms)
      ✓ should let the token create auctions for you
      ✓ shouldn't let anyone create auctions for you
      ✓ shouldn't let you bid if you're not whitelisted (70ms)
      ✓ should let you bid (101ms)
      ✓ should let you outbid (132ms)
      ✓ should let time pass
      ✓ should let you complete auctions (40ms)
      ✓ should complete auctions where no one bids (210ms)
      ✓ should complete auctions where no one bids and the seller is no longer whitelisted (83ms)
      ✓ should complete auctions where no one bids, not whitelisted, and burn doesn't exist
      ✓ should let you withdraw (39ms)

    DEXRouter
      ✓ should be able to place buy orders
      ✓ should use the msg.sender as the trader

    DistributionERC20
      ✓ should not allow delegation
      ✓ should have the correct initial vote/balance value
      ✓ should correctly update on transfer (65ms)
      ✓ should ban fee on transfer
      ✓ should handle distributions (71ms)
      ✓ shouldn't allow claiming multiple times
      ✓ should handle distributions to multiple parties (199ms)
      ✓ should handle distributions to variable parties

    FrabricERC20
      ✓ should have initialized properly (63ms)
      ✓ should allow minting
      ✓ should not allow minting to exceed uint112
      ✓ shouldn't allow transfers to non-whitelisted people
      ✓ should handle whitelisting
      ✓ should allow transferring (38ms)
      ✓ should allow burning
      ✓ should allow freezing and then shouldn't allow them to transfer (114ms)
      ✓ should successfully unfreeze when enough time passes
      ✓ shouldn't allow transferring locked tokens (76ms)
      ✓ should allow transferring tokens which aren't locked even if you have a locked balance
      ✓ shouldn't bother removing if they don't have anything to remove
      ✓ shouldn't still allow removing even if they don't have anything to remove if explicit
      ✓ should allow removing (405ms)
      ✓ should handle setting a new parent
      ✓ should allow freezing if the parent froze (43ms)
      ✓ should allow removing if the parent removed (236ms)
      ✓ should allow pausing and then shouldn't allow transfers (48ms)

    FrabricWhitelist
      ✓ should require any parent implements IFrabricWhitelistCore (61ms)
      ✓ should let you set a parent
      ✓ should track whitelisted
      ✓ should track KYC (39ms)
      ✓ should support updating KYC hashes
      ✓ should nonce KYC hashes
      ✓ should handle removals (38ms)
      ✓ should carry parent whitelisting
      ✓ should carry parent status
      ✓ should considered removed even if parent whitelisted
      ✓ should support going global

    IntegratedLimitOrderDEX
      ✓ should be able to convert values to atomic units properly
      ✓ shouldn't let you sell more than you have (39ms)
      ✓ should be able to place sell orders (103ms)
      ✓ should be able to place buy orders (103ms)
      ✓ should be able to fill sell orders (133ms)
      ✓ should be able to fill buy orders (157ms)
      ✓ should be able to place multiple orders at the same price point (393ms)
      ✓ should be able to cancel sell orders (171ms)
      ✓ should be able to cancel multiple orders at the same price point (126ms)
      ✓ should be able to cancel buy orders (241ms)
      ✓ should be able to partially fill orders (287ms)
      ✓ should be able to fill orders and place a new one (156ms)
      ✓ should allow withdrawing
      ✓ shouldn't allow frozen accounts to place orders (72ms)
      ✓ should let you fill orders of frozen accounts (252ms)
      ✓ should call remove when filling orders (removed) (396ms)
      ✓ should call remove when cancelling orders without a swap remove (removed) (379ms)
      ✓ should call remove when cancelling orders with a swap remove (removed) (528ms)
      ✓ should call remove when filling orders (removing) (376ms)
      ✓ should call remove when cancelling orders without a swap remove (removing) (348ms)
      ✓ should call remove when cancelling orders with a swap remove (removing) (362ms)
      ✓ should call remove when filling orders (removed, dropping buy) (333ms)
      ✓ should call remove when cancelling orders without a swap remove (removed, dropping buy) (307ms)
      ✓ should call remove when cancelling orders with a swap remove (removed, dropping buy) (398ms)
      ✓ should fuzz without issue (16085ms)

    Timelock
      ✓ should allow recovering tokens accidentally sent (60ms)
      ✓ shouldn't let anyone set a lock
      ✓ should let the owner set a lock (42ms)
      ✓ shouldn't allow claiming before the lock expires
      ✓ should allow claiming when the lock expires (275ms)

    Bond
      ✓ shouldn't let non-active-governors add bond
      ✓ should let active governors add bond (90ms)
      ✓ shouldn't let you call unbond
      ✓ shouldn't let you call slash
      ✓ should let the Frabric call unbond (45ms)
      ✓ should let the Frabric call slash (53ms)
      ✓ shouldn't let you recover the bond token
      ✓ should let you recover arbitrary tokens
      ✓ shouldn't let you transfer the bond token

    Frabric
Nothing to compile
      ✓ shouldn't let anyone propose
      ✓ shouldn't let you propose Null/Removed/Genesis participants
      ✓ should let you add KYC agencies (175ms)
      ✓ should let you add participants (134ms)
      ✓ should let you add a Governor (200ms)
      ✓ should let you add a Voucher (182ms)
      ✓ should let governors add bond (382ms)
      ✓ should let you remove bond (211ms)
      ✓ should let you slash bond (188ms)
      ✓ should let you create a Thread (485ms)
      ✓ should let you propose creating a proposal on a Thread (181ms)
      ✓ should correctly handle participant removals (159ms)
      ✓ should let you sell the tokens from a leaving Thread
```

```
InitialFrabric
   ✓ should have initialized correctly (43ms)

Crowdfund
   ✓ should allow depositing (55ms)
   ✓ shouldn't allow people who aren't whitelisted to participate
   ✓ should allow withdrawing
   ✓ shouldn't allow anyone to cancel
   ✓ should allow cancelling
   ✓ shouldn't allow depositing when cancelled
   ✓ should reach target (58ms)
   ✓ shouldn't allow depositing more than the target
   ✓ should only allow the governor to execute
   ✓ should allow executing once it reaches target
   ✓ shouldn't allow depositing when executing
   ✓ should allow finishing
   ✓ shouldn't allow depositing when finished
   ✓ should allow claiming Thread tokens (62ms)
   ✓ should only allow the governor to refund
   ✓ should allow refunding (60ms)

Thread
   ✓ shouldn't let anyone propose
   ✓ should let the Frabric and Governor propose
   ✓ should let whitelisted token holders propose (113ms)
   ✓ shouldn't let you upgrade to an independent piece of code before leaving (62ms)
   ✓ should let you upgrade to the current piece of code (198ms)
   ✓ should't let you remove the Frabric
   ✓ should allow changing the descriptor (150ms)
   ✓ should allow changing the Frabric (295ms)
   ✓ should allow changing the Governor (185ms)
   ✓ should allow leaving the ecosystem (446ms)
   ✓ should allow dissolving (375ms)


186 passing (49s)
```

Code Coverage

Quantstamp usually recommends developers to increase the branch coverage to `90%` and above before a project goes live, in order to avoid hidden functional bugs that might not be easy to spot during the development phase.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
`beacon/`	100	79.17	100	88.89	
`Beacon.sol`	100	77.27	100	88.1	… 119,123,132
`SingleBeacon.sol`	100	100	100	100	
`common/`	100	100	100	100	
`Composable.sol`	100	100	100	100	
`dao/`	97.22	78.95	96.43	88.68	
`DAO.sol`	95.54	82.14	100	92.48	… 284,326,353
`FrabricDAO.sol`	100	75.86	87.5	82.28	… 253,287,341
`erc20/`	96.71	78.85	100	90.96	
`Auction.sol`	97.06	73.33	100	89.74	… 197,200,217
`DEXRouter.sol`	100	100	100	100	
`DistributionERC20.sol`	100	70	100	90.63	72,99,117
`FrabricERC20.sol`	97.1	86.67	100	94.81	115,130,239,278
`FrabricWhitelist.sol`	94.87	70.83	100	85.11	… ,88,109,117
`IntegratedLimitOrderDEX.sol`	98.1	86.54	100	92.92	… 270,281,316
`Timelock.sol`	86.96	60	100	84	41,56,57,58
`frabric/`	98.54	63.89	96	84.94	
`Bond.sol`	100	100	100	100	
`Frabric.sol`	98.06	60.61	100	81.25	… 339,361,381
`InitialFrabric.sol`	100	100	75	87.5	36
`interfaces/beacon/`	100	100	100	100	
`IFrabricBeacon.sol`	100	100	100	100	
`interfaces/common/`	100	100	100	100	
`Errors.sol`	100	100	100	100	
`IComposable.sol`	100	100	100	100	
`IUpgradeable.sol`	100	100	100	100	
`interfaces/dao/`	100	100	100	100	
`IDAO.sol`	100	100	100	100	
`IFrabricDAO.sol`	100	100	100	100	
`interfaces/erc20/`	**100**	**100**	**100**	**100**	
`IAuction.sol`	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
IDEXRouter.sol	100	100	100	100	
IDistributionERC20.sol	100	100	100	100	
IFrabricERC20.sol	100	100	100	100	
IFrabricWhitelist.sol	100	100	100	100	
IIntegratedLimitOrderDEX.sol	100	100	100	100	
ITimelock.sol	100	100	100	100	
interfaces/frabric/	100	100	100	100	
IBond.sol	100	100	100	100	
IFrabric.sol	100	100	100	100	
IInitialFrabric.sol	100	100	100	100	
interfaces/thread/	100	100	100	100	
ICrowdfund.sol	100	100	100	100	
IThread.sol	100	100	100	100	
IThreadDeployer.sol	100	100	100	100	
thread/	96.15	65.48	94.59	85.52	
Crowdfund.sol	95.65	61.36	100	80.68	… 243,252,256
Thread.sol	98.48	73.53	100	89.16	… 241,301,309
ThreadDeployer.sol	93.62	50	75	88	… 231,237,238
All files	**97.16**	**74.12**	**97.71**	**88.34**	

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

```
75108b6073289a1f9189bd2067e2306681b7d3236b6e208ee0e3d2b3a8930fdf ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/beacon/Beacon.sol

96fcd68d5494f8188d75b38e56ca0e8070b6ae186a3fe278b755124957364cd3 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/beacon/SingleBeacon.sol

1b46069cc9cdb900957146ce5022aa7a4697bb9f98250d6668540d2e98c40b27 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/beacon/IFrabricBeacon.sol

a9cfab45b7d38ac258935c1dbae27257d6e350b61a6721764fb750d857c85377 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/thread/IThreadDeployer.sol

5f647e37acda88fe2d1044751661120bc175e2620dbc2bf3da7d01338c0c6ea0 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/thread/ICrowdfund.sol

541c55e6bd99d8b6c3a13ee12d8501fe7ed1104f88e7e686e11ed86ab58754f6 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/thread/IThread.sol

bf977c737aeafd5411f3555ab5b9c420f69a8e493084304f9cd5b25ba79f672b ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/common/Errors.sol

97615121079a85f0c9bd2717f1e6a67b7a3c5b6edba1a66589db8a15118809da ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/common/IUpgradeable.sol

22bbf220e911f7b77ce5160c46ecf684ee7eea68aa7a27063ebeb69612f06b25 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/common/IComposable.sol

854cc8a9ea07a95f0111728624a7af9d05135fc3d1c1a5d7db7710a0e0c65b3e ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/dao/IDAO.sol

b29da6a6f9ffe7d781b2a6bc6cc6f71438eca4af5a5ff5e1bf7751c6b0838b1f ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/dao/IFrabricDAO.sol

cab2e403f7c32133e4b35af64d7b52409a53676e304ffd671c5fa7b8d770454a ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/frabric/IBond.sol

f8c0c75b108c15153ec164beae662c5d1aee6bab41b2ed74d9307f461f46f4c9 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/frabric/IInitialFrabric.sol

5295ea819a409d47c487d7d515e1893320b2a2a86fdfeef304ec7e45e10f558a ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/frabric/IFrabric.sol

678ae1f9e458466addfa404f0aa109728339e8acbb4cd23e6220593e2bda1ab4 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/erc20/IFrabricWhitelist.sol

2ba05f08b46ed67bb6616d49c33859a4449e1929f1f32070582c0dbabb56aa03 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/erc20/IFrabricERC20.sol

2b35edcd9f0d776e964be0ffdd20a9055585585f8cff43a97bc7a5ecb39a8191 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/erc20/IDistributionERC20.sol

cc46543229e9bf6fc148ce580921360959b7b578c1efd06af96c47378ff53801 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/erc20/IIntegratedLimitOrderDEX.sol

8b35aed11d5ea0f52312cd7fd6057c3898f978705d3c2de8cfecee1cf1c3fb41 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/erc20/IAuction.sol

844a10bba585f3fe6f711f1920dd60d92abbb43624ee6373895debbb4387d06a ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/erc20/ITimelock.sol

bb200736c190a740b9e40602db0b117b331eb19b31840b8d1eb021bc51667a21 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/interfaces/erc20/IDEXRouter.sol

3a95eb156e7bc9a75cc58ffb26ca4b3ed95a43266c9aa385d4e3147d9c55a378 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/thread/Crowdfund.sol

dcf2d15d01d7b25acdc60ca2640df1abc6ef6f46dd52f9857fb209913d7d5708 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/thread/ThreadDeployer.sol

1287324ece58747adb4c186c8514c62eda7a0a007889524b1f6094ab6d6f91eb ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/thread/Thread.sol

52516f6f9da3a7fe38b8f068f4a4d2c3cc893667156f27df75478d230de9f4ed ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/common/Composable.sol

520f81e047b3a2e9a91055f97cb88354ceaef94d37cf664415b992159c6860db ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/dao/FrabricDAO.sol

7c5abb05e2bb9c36c87eb69c447cd23c41233565e06bdd3f9d2a0a8c7bf15a45 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/dao/DAO.sol

b793fa4b4077cc79127b1365e2ce644aa8ebf511dd9ac08c5d746052455ad050 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/frabric/InitialFrabric.sol

f38413bf13e9cc3c339d3b4dd8e4e766a2b7e7b1939da44abf09f50034ca1d46 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/frabric/Frabric.sol

2e947690df951f3eedcaa0d0ee8cd6a0bd2fd30523610e8a27b05c88d4a7abed ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/frabric/Bond.sol

163f88aa2a2855c39a35aed49dba122408157a1f81aa73b774338e00843815e5 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/erc20/Timelock.sol

df0971328638576af7eb40c30da3ce744e1ad83adbf3d1eaab5282e031d984c3 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/erc20/FrabricERC20.sol

56ea93e4e11f30238449dfff74fa8b290e521795157f4c315d187b07fa2fbb30 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/erc20/IntegratedLimitOrderDEX.sol

99e5c2a61d865f429d8ea475781072f55f07512d9d045eea51b187e7469b1931 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/erc20/DEXRouter.sol

1dd5d896db09374738876d2a6e9b8f778990a43be22433dcf50b3e0c873ca1ea ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/erc20/Auction.sol

71f656e36d29b5638f419f4f26f8a69ee34f0ff2110a8b4cbf1c155b5f1fe61f ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/erc20/DistributionERC20.sol

a22e03df20597042d94c742373bc8d67c8a7bb7e75477313d56063fb79d94ef8 ./fractional_finance-frabric-protocol-fcff8a555d1b-
github/contracts/erc20/FrabricWhitelist.sol
```

Changelog

- 2022-06-09 - Initial report
- 2022-07-01 - re-audit 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.