Quantstamp Security Assessment Certificate

April 6th 2021 — Quantstamp Verified

Cryptex Finance

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

Executive Summary

Type	Collateralized asset
Auditors	Sebastian Banescu, Senior Research Engineer Jose Ignacio Orlicki, Senior Engineer Joseph Xu, Technical R&D Advisor
Timeline	2021-03-01 through 2021-03-18
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	TCAP Documentation Whitepaper
Documentation Quality	Medium
Test Quality	Undetermined
Source Code	

Repository	Commit
contracts	9bd0481...755d32e

Total Issues	23	(21 Resolved)
High Risk Issues	1	(1 Resolved)
Medium Risk Issues	2	(2 Resolved)
Low Risk Issues	8	(8 Resolved)
Informational Risk Issues	8	(7 Resolved)
Undetermined Risk Issues	4	(3 Resolved)

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

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

Summary of Findings

Quantstamp has performed an audit of the diff corresponding to the commit hash `9bd0481` which was previously audited and including changes up to commit hash `755d32e`. The changes mainly involve newly added code for the project governance. During this audit we have identified 23 security issues ranging through all security levels, 4 issues in the specification, 2 issues in code comments and 3 deviations from best practices. Additionally we have noticed 3 failing tests which lead the coverage to be shown as 0% for the newly added governance contracts. We recommend addressing all these issues before deploying the smart contracts in production.

ID	Description	Severity	Status
QSP-1	Wrong token being transferred on `claimVest()`	High	Fixed
QSP-2	Staking token can be withdrawn from the rewards contract	Medium	Fixed
QSP-3	TCAP Token can be withdrawn from the vault	Medium	Mitigated
QSP-4	Oracle price could be stale	Low	Fixed
QSP-5	Loss of rewards due to truncation	Low	Fixed
QSP-6	Ratio can be set to any value	Low	Fixed
QSP-7	Burn fee can be set to any value	Low	Fixed
QSP-8	Treasury can be set to any address	Low	Fixed
QSP-9	Burn fee not paid to the treasury	Low	Fixed
QSP-10	Median is more robust than average for aggregated oracles	Low	Fixed
QSP-11	Dangerous use of strict equality	Low	Fixed
QSP-12	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-13	Misleading error message	Informational	Fixed
QSP-14	Single point of failure for price feeds	Informational	Acknowledged
QSP-15	Clone-and-Own	Informational	Fixed
QSP-16	Receipts with value zero for invalid proposals	Informational	Fixed
QSP-17	Loss of precision due to multiplication after division	Informational	Fixed
QSP-18	Unchecked Return Value	Informational	Fixed
QSP-19	Missing input `address` validation	Informational	Fixed
QSP-20	Incorrect amount may be withdrawn from the reward handler	Undetermined	Acknowledged
QSP-21	`vestingBegin` state variable not read in the `LiquidityReward` contract	Undetermined	Fixed
QSP-22	Unclear vesting period interpretation	Undetermined	Fixed
QSP-23	`vestingRatio` can be arbitrarily set in the `LiquidityReward` constructor	Undetermined	Fixed

Quantstamp Audit Breakdown

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

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

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

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Toolset

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

Setup

Tool Setup:

- Slither v0.7.0

Steps taken to run the tools:

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

Findings

QSP-1 Wrong token being transferred on `claimVest()`

Severity: High Risk

Status: Fixed

File(s) affected: `LiquidityReward.sol`

Description: The `claimVest()` function transfers the staking token as opposed to the reward token.

Recommendation: Transfer the reward token instead of the staking token on L183.

QSP-2 Staking token can be withdrawn from the rewards contract

Severity: Medium Risk

Status: Fixed

File(s) affected: `LiquidityReward.sol`

Description: The `LiquidityReward.recoverERC20()` function contains a check at the start, which only checks that the `_tokenAddress` of the ERC20 token to withdraw is not equal to the `rewardsToken`. However, the error message of this check states that: "LiquidityReward::recoverERC20: Cannot withdraw the staking or rewards tokens". Note that even though the error message indicates that the staking token cannot with withdrawn, there is no check to prevent this.

Recommendation: Add a conjunction that also checks `_tokenAddress != address(stakingToken)` in the same `require` statement.

QSP-3 TCAP Token can be withdrawn from the vault

Severity: Medium Risk

Status: Mitigated

File(s) affected: `IVaultHandler.sol`

Description: The owner can recover both TCAP Token and any ERC20 collateral using `recoverERC20()` (unless TCAP Token is the collateral).

Recommendation: Change `||` in Line 541 to `&&`.

Update: The specification and code have been updated: the TCAP token can still be withdrawn. The collateral token cannot be withdrawn anymore.

QSP-4 Oracle price could be stale

Severity: Low Risk

Status: Fixed

File(s) affected: `ChainlinkOracle.sol`

Description: The `ChainlinkOracle.getLatestAnswer()` function simply returns the price from the call to `AggregatorV3Interface.latestRoundData()` of Chainlink, ignoring all other return values from this function. This could lead to stale prices according to the Chainlink documentation:

1. under current notifications: "if answeredInRound < roundId could indicate stale data."
2. under historical price data: "A timestamp with zero value means the round is not complete and should not be used."

Recommendation: We recommend adding require statements that check for the aforementioned conditions in all the occurrences of those functions.

QSP-5 Loss of rewards due to truncation

Severity: Low Risk

Status: Fixed

File(s) affected: `LiquidityReward.sol`

Description: The `LiquidityReward.getReward()` function splits the `reward` into `vestingReward` and `transferReward`. However, it does this in an inefficient and imprecise way, which could lead to small reward losses due to the truncation caused by integer division. The following code snippet is used:

```
uint256 hundred = 100;
uint256 vestingReward = (reward.mul(vestingRatio)).div(100);
uint256 transferReward = (reward.mul(hundred.sub(vestingRatio))).div(100);
```

Recommendation: Compute `transferReward` based on the value of `vestingReward`, which would make the code more efficient and eliminate any losses:

```
uint256 vestingReward = (reward.mul(vestingRatio)).div(100);
uint256 transferReward = reward.sub(vestingReward);
```

QSP-6 Ratio can be set to any value

Severity: Low Risk

Status: Fixed

File(s) affected: `IVaultHandler.sol`

Description: The `IVaultHandler.setRatio()` function does not contain any constraints on the value of the `_ratio` input parameter. Therefore, the owner of the contract could set the `ratio` value to any unsigned integer. Setting it to a value lower than 100 could have very serious consequences.

Recommendation: Given that the documentation and whitepaper indicate that TCAP is "150% fully backed", there should be a check that `ratio > 150`.

QSP-7 Burn fee can be set to any value

Severity: Low Risk

Status: Fixed

File(s) affected: `IVaultHandler.sol`

Description: The `IVaultHandler.setBurnFee()` function does not contain any constraints on the value of the `_burnFee` input parameter. Therefore, the owner of the contract could set the `burnFee` value to any unsigned integer. Setting it to a value close-to or grater than 100 could have very serious consequences.

Recommendation: Decide on a maximum acceptable value for the `burnFee` and add a require statement that checks that the `burnFee` is never set above this limit. The relationship between the burn fee and liquidation penalty should also be taken into consideration because the burn fee should be significantly lower than the liquidation penalty to incentivize keepers.

QSP-8 Treasury can be set to any address

Severity: Low Risk

Status: Fixed

File(s) affected: `IVaultHandler.sol`

Description: The `IVaultHandler.setTreasury()` function does not contain any constraints on the value of the `_treasury` input parameter. Therefore, the owner of the contract could set the `treasury` value to any address including an EOA.

Recommendation: Add an interface for the treasury contract. The treasury should have an interface constant that conforms to ERC165, which can be checked using the `ERC165Checker.supportsInterface()` function. This would also make treasury management more transparent for end-users.

QSP-9 Burn fee not paid to the treasury

Severity: *Low Risk*

Status: Fixed

File(s) affected: `IVaultHandler.sol`

Description: The burn fee is not paid to the treasury in the `liquidateVault()` function. Comments indicate that this is currently a //TODO item.

Recommendation: Complete the //TODO item.

QSP-10 Median is more robust than average for aggregated oracles

Severity: *Low Risk*

Status: Fixed

Description: The whitepaper shows an example of 5 data sources for TCAP. If 1 or 2 of these 5 data sources are compromised and start sending irregular prices, using a median price aggregation of total market cap sources, instead of an average price, is more robust. This allows the aggregated price to continue to operate without pause until the sources are fixed or decommissioned. Otherwise, the incident must be immediately detected and trading stopped until source data is fixed or decommissioned.

Exploit Scenario:

1. Mallory compromises the CoinExample data source.
2. CoinExample starts sending a price with a 50% premium above the previous average.
3. On average, the TCAP price goes up by 10%
4. Mallory burns TCAP tokens getting a 10% premium.

Recommendation: Use median instead of average (mean) to aggregate the oracle prices.

Update: The Chainlink oracle used in the implementation provides the median value by default. The team has updated the whitepaper such that it specifies the median instead of the average value.

QSP-11 Dangerous use of strict equality

Severity: *Low Risk*

Status: Fixed

File(s) affected: `IVaultHandler.sol`

Description: Two different instances of this issue have been found:

1. The `IVaultHandler.liquidateVault()` function makes use of a strict equality between the return value of `requiredLiquidationTCAP(vault.Id)` and the `_requiredTCAP` input parameter. This puts an unnecessary burden on the caller to compute the required amount of TCAP for the liquidation and in case the price is moving fast (in either direction), the call to `liquidateVault()` might fail even though the caller is willing to pay up to a certain amount of TCAP tokens.
2. The `IVaultHandler.withBurnFee()` modifier makes use of strict equality between the `msg.value` and the `fee`. This puts an unnecessary burden on the caller of the `burn()` function to compute the exact fee amount. Otherwise, the call will fail.

Recommendation: In the order of the items above:

1. Change the design of this function such that `liquidateVault()` takes in the maximum amount of TCAP (`_maxTCAP`) that the caller is willing to pay, instead of the exact amount necessary for liquidation (`_requiredTCAP`). Also change the `require` condition on L476 from strict equality to `_maxTCAP >= req`.
2. Change the strict equality in the `IVaultHandler.withBurnFee()` modifier to `fee <= msg.value` and return the excess amount back to the caller.

QSP-12 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Mitigated

File(s) affected: `Ctx.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens. This issue is already acknowledged in the comments of the `approve()` function.

Exploit Scenario:

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.

Recommendation: 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.

Update: The exploit (as described above) is mitigated through the use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`.

QSP-13 Misleading error message

Severity: *Informational*

Status: Fixed

File(s) affected: `IVaultHandler.sol`

Description: The `IVaultHandler.recoverERC20()` function allows the `IVaultHandler` contract owner to recover "LP Rewards from other systems such as BAL to be distributed to holders". According to the error message in the single `require` statement from this function: "Cannot withdraw the staking, collateral or rewards tokens". However, only the staking and collateral token addresses are checked by that `require` statement. It is unclear what the "rewards tokens" part of the error message is referring to, however, the one thing that comes to mind is CTX tokens, which are never supposed to reach the `IVaultHandler` contract.

Recommendation: Either change the error message in the `require` statement to indicate that only the staking and collateral tokens cannot be withdrawn or in case CTX tokens would be held by the `IVaultHandler` contract, add another disjunction to the condition checked by the `require` statement, which should check that the `_tokenAddress` is different from the CTX token address.

Update: Removed the TCAP Token from the `require` statement and updated the error message to indicate that only collateral should be recoverable. This issue is related to QSP-3.

QSP-14 Single point of failure for price feeds

Severity: *Informational*

Status: Acknowledged

File(s) affected: `IVaultHandler.sol`

Description: The price feeds rely on a single oracle, namely the Chainlink Aggregator V3, which is indeed robust. However, in the event of any large scale attack/disruption of the Chainlink network, the Cryptex vault handlers would be severely impacted.

Recommendation: Consider adding at least one other robust price feed, which is independent of Chainlink.

Update: From dev team:

> For the first months Chainlink will be the default oracle. We will upgrade to a more robust version in the future with the support of different oracles. We added some functions to increase protection from users like a max cap on TCAP supply that can be updated. Pausing minting of TCAP can help control the situation in case Chainlink goes down.

QSP-15 Clone-and-Own

Severity: *Informational*

Status: Fixed

File(s) affected: `SafeMath.sol`

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

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

QSP-16 Receipts with value zero for invalid proposals

Severity: *Informational*

Status: Fixed

File(s) affected: `GovernorAlpha.sol`

Description: The `getReceipt()` function can return receipts with value zero for invalid proposals because `mapping(address => Receipt) receipts` got moved out of the `Proposal` struct. It is not clear why this change was introduced since the prior audit, as everything else is either naming or linting changes.

Recommendation: Add the following statement `require(proposalCount >= proposalId && proposalId > 0, "GovernorAlpha::getReceipt: invalid proposal id");`.

QSP-17 Loss of precision due to multiplication after division

Severity: *Informational*

Status: Fixed

File(s) affected: `IVaultHandler.sol`

Description: Division in Solidity leads to truncation and loss of precision. The effect of truncation is exacerbated if multiplication is performed on the result of a division. The following instance has been detected:

1. `IVaultHandler.liquidationReward` performs a multiplication on the result of a division:
   - `reward = (req.mul(liquidationPenalty.add(100))).div(100)` (contracts/IVaultHandler.sol#684)
   - `rewardCollateral = (reward.mul(tcapPrice)).div(collateralPrice)` (contracts/IVaultHandler.sol#685)

Recommendation: Try to perform division after multiplication or describe why this is not possible in the 3 items from the description.

QSP-18 Unchecked Return Value

Severity: Informational

Status: Fixed

File(s) affected: `LiquidityReward.sol`

Description: Most functions will return a `true` or `false` value upon success. Some functions, like `send()`, are more crucial to check than others. It's important to ensure that every necessary function is checked. `LiquidityReward.claimVest()` L176-184, ignores return value by `stakingToken.transfer()`. If the `transfer()` function of the staking token does not fail on an unsuccessful transfer, then the `claimVest()` function would return successfully without the sender having received the claim amount.

Recommendation: Check the return value of `transfer()` on L183.

Update: Fixed by using `safeTransfer()` instead of `transfer()` on L183.

QSP-19 Missing input `address` validation

Severity: Informational

Status: Fixed

File(s) affected: `Orchestrator.sol`, `RewardHandler.sol`

Description: Input parameters of type `address` should always be checked to be different from `address(0)` to avoid sending funds to such an address by mistake. The following functions and parameters are lacking such a check:

1. `Orchestrator.constructor(address)._guardian` in `Orchestrator.sol` on L96.
2. `Orchestrator.retrieveETH(address)._to` in `Orchestrator.sol` on L247.
3. `Orchestrator.executeTransaction(address,uint256,string,bytes).target` lacks a zero-check on `(success,returnData) = target.call{value: value}(callData)` on L328-329.
4. `RewardHandler.constructor(address,address,address)._vault` in `RewardHandler.sol` on L92.

Recommendation: Add `require` statements to check that the values enumerated above are not `address(0)` or provide a description as to why this is not needed.

QSP-20 Incorrect amount may be withdrawn from the reward handler

Severity: Undetermined

Status: Acknowledged

File(s) affected: `IVaultHandler.sol`

Description: The `IVaultHandler.liquidateVault()` function withdraws `_requiredTCAP` from the `rewardHandler` for the `vault.Owner` account. However, this amount may be larger than the amount that was staked when the owner of the vault minted TCAP, because the `_requiredTCAP` is the output of `requiredLiquidationTCAP()`, which is different than the vault debt. Note that the vault owner has only staked an amount equal to the amount of TCAP tokens minted, which is equal to the vault debt. If the `_requiredTCAP > vault.Debt`, then the `liquidateVault()` call will fail due to an integer underflow that is caught by the SafeMath `sub` function inside of the `RewardHandler.withdraw()` function.

Recommendation: Withdraw `vault.Debt` instead of `_requiredTCAP` on L489.

Update: From dev team:

> `RewardHandler` tracks the current debt of the user, removing all the debt will leave a false value as TCAP debt still exists from the user. A liquidation doesn't remove all debt, only sets the vault back to a safer ratio, meaning that `requiredTCAP` won't be `>` than `Debt`.

QSP-21 `vestingBegin` state variable not read in the `LiquidityReward` contract

Severity: Undetermined

Status: Fixed

File(s) affected: `LiquidityReward.sol`

Description: On the one hand the `vestingBegin` state variable is never read in the `LiquidityReward` contract, which may indicate that it is not actually needed. On the other hand, the whitepaper indicates that the vesting period should be 6 months:

> In order to minimize the volatility of CTX due to new issuance from community rewards, newly issued CTX tokens shall be subjected to a vesting period of 6 months where 1/3rd of the reward is immediately available while the remaining 2/3rds reward will not be accessible until 6 months vesting period has been reached.

However, the code allows setting the `vestingBegin` and `vestingEnd` arbitrarily and hence the vesting period could be different than 6 months.

Recommendation: Either enforce that the vesting period be 6 months in the code, or update the specification to reflect that the vesting period could be arbitrarily determined by the contract deployer.

Update: From dev team:

> Removed the `vestingBegin` variable as it's not needed. The deployer can define the duration of the rewards using the `vestingEnds` variable.

QSP-22 Unclear vesting period interpretation

Severity: Undetermined

Status: Fixed

File(s) affected: `LiquidityReward.sol`

Description: The whitepaper indicates that:

> In order to minimize the volatility of CTX due to new issuance from community rewards, newly issued CTX tokens shall be subjected to a vesting period of 6 months where 1/3rd of the reward is immediately available while the remaining 2/3rds reward will not be accessible until 6 months vesting period has been reached.

This can be interpreted in 2 ways:

1. Whenever a user calls `getRewards()` a new 6 month vesting period starts for the rewards of that user.
2. There is a global vesting period of 6 months and once that period ends any call to `getRewards()` will allow the user to obtain the full reward.

The implementation in `LiquidityReward` is essentially the 2nd option, however, with the inefficiency that the user needs to also call `claimVest()` after calling `getRewards()` in order to obtain the full reward after the global vesting period has ended.

Recommendation: If the 2nd option is indeed intended, then add a check in `getReward()` and allow the users to obtain the full reward after the global vesting period has ended, without needing to call `claimVest()`. Otherwise, if the 1st option is intended the `vestingBegin` and `vestingEnd` state variables must be turned into mappings that hold individual vesting periods.

Update: Fixed according to option 2. From dev team:

> Added transfer of all the rewards if time is greater than vesting.

QSP-23 `vestingRatio` can be arbitrarily set in the `LiquidityReward` constructor

Severity: *Undetermined*

Status: Fixed

File(s) affected: `LiquidityReward.sol`

Description: One the one hand, the `vestingRatio` can be arbitrarily set in the `LiquidityReward` constructor, because there is no constraint on the value of the state variable being between 0 and 100%. On the other hand, the code comment of the `LiquidityReward.getReward()` function says: "only 70% of reward is inmediate transfered the rest is locked into vesting". Moreover, the whitepaper indicates something slightly different:

> In order to minimize the volatility of CTX due to new issuance from community rewards, newly issued CTX tokens shall be subjected to a vesting period of 6 months where 1/3rd of the reward is immediately available while the remaining 2/3rds reward will not be accessible until 6 months vesting period has been reached.

Recommendation: Align the code comments and the whitepaper to the right amount. Since this amount seems to be fixed, it should be a constant rather than a state variable that is initialized in the constructor. In addition, modify the comment for `getReward()` function to say "30% of reward is vested and the rest immediately transferred". The current @dev comment can cause confusion and future developers may incorrectly set `vestingRatio = 70`.

Update: From dev team:

> Updated comments and whitepaper, for the vesting ratio, also the variable is set on the constructor as the reward might change for different reward programs in the future

Automated Analyses

Slither

Slither has detected 225 results out of which the majority have been filtered out as false positives and the rest have been integrated in the findings from this report.

Adherence to Specification

The following deviations from the specification were encountered:

1. The Orchestrator description says that: "The Orchestrator implements a 3 day timelock for each function in order to change the configuration settings of it's child components [...]". This is not the case for any of the `Orchestrtor` contract functions. This timelocked feature that the text is referring to seems to have been added to the `GovernorAlpha` contract. However, we did not find any description for the `GovernorAlpha` contract in the specification. Moreover, the 3 day timelock seems to be only an example value, because the `Timelock` contract allows delays between 2 and 30 days. Users will have to check actual `delay` value set in the `Timelock` contract instance used by `GovernorAlpha`.
2. The dedicated Orchestrator page indicates that the `Orchestrator` contract has a `notLocked` modifier which was not found in the code.
3. Both the whitepaper and the docs website say that TCAP is a "150% fully backed, fully collateralized asset". However, this 150% value is not fixed in the code and can be set to any value by the owner/governance that can call the `IVaultHandler.setRatio()` function at any time.
4. There are multiple typos in the whitepaper regarding the calculation methodology, especially under the Liquidation Event section.
   - The denominator for the liquidation TCAP required LA should be $\mathrm{r} - (\mathrm{p} + 100)$.
   - Division is used instead of multiplication for $(\mathrm{C} * \mathrm{cp})/\mathrm{P}$.

Code Documentation

The following issues were encountered in the code comments:

1. **[Unresolved]** Typo on L543 in `IVaultHandler.sol`: "rewards tokens" -> "reward tokens".
2. **[Fixed]** Typo on L300 in `LiquidityRewards.sol`: "inmediate transfered" -> "immediately transferred".

Adherence to Best Practices

The following deviations from best practices have been encoutered during the audit:

1. The following `event` parameters of type `address` are not `indexed`:

    . **[Fixed]** `_rewardHandler` in the `NewRewardHandler` event

    . **[Unresolved]** `_tresury` in the `NewTreasury` event

    . **[Unresolved]** `_token` in the `Recovered` event

2. **[Fixed]** TODOs should not be present in production code. Here are the instances we found:

    . "TODO: this should be modifier" on L736 in `IVaultHandler.sol`

    . "@dev The fee goes to the treasury contract //TODO" on L457 in `IVaultHandler.sol`

    . "TODO: Add Permit for gasless transactions" on L10 in `TCAP.sol`

3. **[Unresolved]** Typo in parameter name on L132 in `IVaultHandler.sol`: `address _tresury`.

Test Results

Test Suite Results

During test execution, we have noticed 3 failing tests. Error messages are given below.
<
Update: The aforementioned issue has been fixed and we confirm that all 108 tests are passing

```
Network Info
============
> HardhatEVM: v2.0.8
> network:    hardhat



  Chainlink Oracle
    ✓ ...should deploy the contract (176ms)
    ✓ ...should get the oracle answer

  ERC20 Vault
    ✓ ...should deploy the contract (1089ms)
    ✓ ...should allow the owner to set the treasury address (160ms)
    ✓ ...should return the token price
    ✓ ...should allow users to create a vault (76ms)
    ✓ ...should get vault by id
    ✓ ...should allow user to stake collateral (294ms)
    ✓ ...should allow user to retrieve unused collateral (214ms)
    ✓ ...should return the correct minimal collateral required (80ms)
    ✓ ...shouldn't allow minting above cap (281ms)
    ✓ ...should allow user to mint tokens (195ms)
    ✓ ...should allow token transfers (62ms)
    ✓ ...shouldn't allow user to send tokens to tcap contract
    ✓ ...should allow users to get collateral ratio
    ✓ ...shouln't allow users to retrieve stake unless debt is paid (54ms)
    ✓ ...should calculate the burn fee (66ms)
    ✓ ...should allow users to burn tokens (250ms)
    ✓ ...should update the collateral ratio
    ✓ ...should allow users to retrieve stake when debt is paid (44ms)
    ✓ ...should test liquidation requirements (194ms)
    ✓ ...should get the required collateral for liquidation (68ms)
    ✓ ...should get the liquidation reward (94ms)
    ✓ ...should allow liquidators to return profits (73ms)
    ✓ ...should allow users to liquidate users on vault ratio less than ratio (584ms)
    ✓ ...should allow owner to pause contract (47ms)
    ✓ ...shouldn't allow contract calls if contract is paused (40ms)
    ✓ ...should allow owner to unpause contract (42ms)

  ETH Vault
    ✓ ...should deploy the contract (1088ms)
    ✓ ...should allow the owner to set the treasury address (125ms)
    ✓ ...should return the token price
    ✓ ...should allow users to create a vault (58ms)
    ✓ ...should get vault by id
    ✓ ...should allow user to stake weth collateral (200ms)
    ✓ ...should allow user to stake eth collateral (117ms)
    ✓ ...should allow user to retrieve unused collateral on eth (152ms)
    ✓ ...should allow user to retrieve unused collateral on weth (159ms)
    ✓ ...should return the correct minimal collateral required (53ms)
    ✓ ...should allow to earn fees if reward address is set (65ms)
    ✓ ...should allow user to mint tokens (288ms)
    ✓ ...should allow user to earn rewards
    ✓ ...should allow users to get collateral ratio
    ✓ ...shouln't allow users to retrieve stake unless debt is paid (56ms)
    ✓ ...should calculate the burn fee (65ms)
    ✓ ...should allow users to burn tokens (308ms)
    ✓ ...should update the collateral ratio
    ✓ ...should allow users to retrieve stake when debt is paid
    ✓ ...should test liquidation requirements (219ms)
    ✓ ...should get the required collateral for liquidation (69ms)
    ✓ ...should get the liquidation reward (95ms)
    ✓ ...should allow liquidators to return profits (83ms)
    ✓ ...should allow users to liquidate users on vault ratio less than ratio (578ms)
    ✓ ...should allow owner to pause contract (45ms)
    ✓ ...shouldn't allow contract calls if contract is paused (39ms)
    ✓ ...should allow owner to unpause contract (45ms)

  Liquidity Mining Reward
    ✓ ...should deploy the contract (263ms)
    ✓ ...should set the constructor values
    ✓ ...should allow an user to stake (105ms)
    ✓ ...should allow owner to fund the reward handler (66ms)
    ✓ ...should allow user to earn rewards (80ms)
    ✓ ...should allow user to retrieve rewards (67ms)
    ✓ ...should allow user to withdraw (77ms)
    ✓ ...should allow vault to exit (144ms)
    ✓ ...shouldn't allow to earn after period finish  (87ms)
    ✓ ...should allow to claim vesting after vesting time (47ms)

  Orchestrator Contract
    ✓ ...should deploy the contract (1318ms)
    ✓ ...should set the owner
    ✓ ...should set the guardian (48ms)
    ✓ ...should set vault ratio (85ms)
    ✓ ...should set vault burn fee (74ms)
    ✓ ...should set vault liquidation penalty (61ms)
    ✓ ...should prevent liquidation penalty + 100 to be above ratio
    ✓ ...should pause the Vault (139ms)
    ✓ ...should unpause the vault (67ms)
    ✓ ...should set the liquidation penalty to 0 on emergency (143ms)
    ✓ ...should set the burn fee to 0 on emergency (134ms)
    ✓ ...should be able to send funds to owner of orchestrator
    ✓ ...should enable the TCAP cap (62ms)
    ✓ ...should set the TCAP cap (61ms)
    ✓ ...should add vault to TCAP token (106ms)
    ✓ ...should remove vault to TCAP token (102ms)
    ✓ ...should allow to execute a custom transaction (124ms)

  Reward Handler
    ✓ ...should deploy the contract (304ms)
    ✓ ...should set the constructor values
    ✓ ...should allow a vault to stake for a user (91ms)
    ✓ ...should allow owner to fund the reward handler (65ms)
    ✓ ...should allow user to earn rewards (84ms)
    ✓ ...should allow user to retrieve rewards (48ms)
    ✓ ...should allow vault to withdraw (67ms)
    ✓ ...should allow vault to exit (103ms)
    ✓ ...shouldn't allow to earn after period finish  (80ms)

  TCAP Token
```

```
    ✓ ...should deploy the contract (206ms)
    ✓ ...should set the correct initial values
    ✓ ...should have the ERC20 standard functions
    ✓ ...should allow to approve tokens
    ✓ ...shouldn't allow users to mint
    ✓ ...shouldn't allow users to burn

  Ctx
    ✓ ...should permit (338ms)
    ✓ ...should changes allowance (557ms)
    ✓ ...should allow nested delegation (386ms)
    ✓ ...should mint (630ms)

  GovernorAlpha
    ✓ ...should test ctx
    ✓ ...should set timelock
    ✓ ...should set governor

  scenario:TreasuryVester
    ✓ setRecipient:fail
    ✓ claim:fail
    ✓ claim:~half (271ms)
    ✓ claim:all (266ms)


  108 passing (17s)
```

Code Coverage

Due to the 3 failing tests, the coverage for the `contracts/governance/` directory is 0% across the board. We recommend fixing the tests and improving coverage such that it is close to 100%.

Update: The issue mentioned above has been fixed and the coverage could be computed for the `contracts/governance/` directory. However, as indicated in the table below the branch coverage is low for these contracts. We recommend adding more tests and increasing the coverage along with adding assertions to check the intended effects and side-effects of each test case.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	93.67	76.32	93.68	93.6	
ERC20VaultHandler.sol	100	100	100	100	
ETHVaultHandler.sol	93.33	70	100	93.75	116
IVaultHandler.sol	97.62	82.26	96.3	97.66	567,571,572
IWETH.sol	100	100	100	100	
LiquidityReward.sol	87.3	68.18	88.24	87.5	… 246,319,320
Orchestrator.sol	94.74	76.92	94.74	93.02	92,99,308
RewardHandler.sol	89.29	66.67	88.89	89.66	… 236,240,241
TCAP.sol	100	87.5	100	100	
contracts/governance/	44.37	25.58	50.88	45.33	
Ctx.sol	74.82	45.95	88	75.54	… 527,530,618
GovernorAlpha.sol	3.23	0	10	3.37	… 441,442,445
Timelock.sol	11.9	6.25	22.22	11.9	… 215,217,222
TreasuryVester.sol	94.74	66.67	100	94.74	57
All files	**69.95**	**49.38**	**77.63**	**70.99**	

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

```
9b9a2a16575e2fe8b20696a5dbd50ffd4874ab7d412e4a3df5ffe2a3f78820e8 ./contracts/ETHVaultHandler.sol
48f2eeed0425d217510330b7ea865c3f3c17669a251222a4f9d980f9d153f0d3 ./contracts/sebi2StakingRewards.sol
3067acf2b02ffac0937b4dc715fb2f0e0092e652ef750efa1685c3f23b6f6df2 ./contracts/IVaultHandler.sol
a2e2f7c7ae8c568179c26c726f029d975e7e4ee5856a52b277b37b2ab736ccd2 ./contracts/Orchestrator.sol
f92b0cb84c05d04da7ef18ce4c8fded70ba99698af073b483c3308c78ae9a004 ./contracts/IWETH.sol
255883931ac26b3518f86c04ccbdee9a12e805bb1fb010d70965e66c380becdd ./contracts/LiquidityReward.sol
8fc90c5c32d7b3e136374371f655195806daab303497af3ff4ecd3a2d8812e2a ./contracts/ERC20VaultHandler.sol
488f09c03f50e830927d04fa9ee6c89306af3e4237984e9eb371a1ec298e2361 ./contracts/TCAP.sol
b896f46d133f4ff3402ce8aa16cd8f2225b2b156f144c4ea930614d689a185da ./contracts/RewardHandler.sol
756897c911ef44bbe39cde828c08e3ada3c8b27d5f8b9f9e803c1920a0ec8449 ./contracts/sebiStakingRewards.sol
d1700f619a9283777ad827bacc773ba4f0964ea41a96a368021fe4f41f367097 ./contracts/governance/TreasuryVester.sol
55df82c54711f986cfc7dc8fb7ede7f5d2ff2c293bc1c09c1abc832e5329c20c ./contracts/governance/Ctx.sol
c73fcff372d8199b723463fbb403c43a7ed99a80dc0ec64a245516492e02e94b ./contracts/governance/sebiSafeMath.sol
090789c17758ce58417b278ed465e44a9dfc3bca793b0da9ce8e2ae1291f7d3c ./contracts/governance/SafeMath.sol
dba4cf826d2d0666774182f6916ef65e37997c692b2a49442b0aeebfe27b8344 ./contracts/governance/GovernorAlpha.sol
ea4204fc8c5c72a5f4984177c209a16be5d538f1a3ee826744c901c21d27e382 ./contracts/governance/sebiTimelock.sol
64376571ef3c7285913859828c5bcb975836f801f792c443725710f6c397d522 ./contracts/governance/Timelock.sol
450d5ad5f47289f11f489211912ae7234ece85c303189df5b7c51babba70848f ./contracts/governance/sebiTreasuryVester.sol
b5b3265263d3591deb5a6c199bfdeaa655be10ab7392c36edd77986d704b9036 ./contracts/governance/sebiGovernorAlpha.sol
2c5e81aece21281888de638d37783cb9eca11649bbdf310e30ca0f8dbc6eb728 ./contracts/governance/sebiUni.sol
ad1633011649fa19c833b3d20d50bcd9549b882527023d2e02509fc75516184a ./contracts/oracles/ChainlinkOracle.sol
f2de9ff17b73ca87497f1fcaad7eadaa973e6598d3cc0dfd9f5107c18fbe8703 ./contracts/mocks/DAI.sol
c87d65a7a86b00ce2802a8c2a22eae84bb5238756f59ef676612568f3f4c82b8 ./contracts/mocks/WBTC.sol
664207d5a162b6e7acc8c4dd7b4c740c2ac784ffafcd12160a57b0081384ad1c ./contracts/mocks/AggregatorInterfaceStable.sol
50fd5349fe3bc0f7f3a0f85910352becd41ca73ed045c1fc7dfda30243ad5edf ./contracts/mocks/AggregatorInterface.sol
bf8447f7052ad48c07f3e031ac332ce06be77281ad80d122796ca76d5dcdd61d ./contracts/mocks/WETH.sol
c548bba59dafde00e99819f12c726b18bfb3ffded80f320d238e99c504e15ac0 ./contracts/mocks/AggregatorInterfaceTCAP.sol
```

Tests

```
51e414e1872d56549cd377461958911750f5e8b129f7eac5dea818cd6d5d4389 ./test/Orchestrator.test.js
45deedc01288cafeb3e23e1581deed79e80db55f2610d4b71618ff9fb335bba8 ./test/ChainlinkOracle.test.js
a5ade862d59f12e33b25928de76fb9cb92e6c99e79736a1a3eaf599adcedc831 ./test/ETHVaultHandler.test.js
273d2cccd2b31692426aac4508c97c0f5b98c2611d3c6c488a32023eba513b1c ./test/TCAP.test.js
8901e462232e7abe44e3483d8564cd7c56ef7a4987acd417a42e4e2b7af0a519 ./test/RewardHandler.test.js
3021ef52d175a7479efdb6e01e840b8c398a0c6596a30ba7e14d6dc6a7196345 ./test/LiquidityRewards.test.js
5ef865242759b22610cdaecfe6bc2b00d4fe5998b5278f2b057d10c00b48b0d2 ./test/ERCVaultHandler.test.js
7cb2113a1ae5fb5bc6df08b67979b7969ff56663ed8e2339ee5688ed5babb582 ./test/governance/fixtures.ts
7c6f161a8455eec6c7e4e3b8f446561d9a28480ced5059c52b53113dbd505406 ./test/governance/utils.ts
63ea0130bdc834fb3b7830c759f254a36e4db4ee1bccafc3f80aa5d404a72768 ./test/governance/Ctx.test.ts
30e743ea02b8d7b923f5a95d991b9f286876c714395162ccada9cdf4eaca3f6d ./test/governance/TreasuryVester.test.ts
a854277b2b679ced8d1aa7eccf2d1b7caa90c6b1a0626f19018dedaddae0ae5e ./test/governance/GovernorAlpha.test.ts
```

Changelog

- 2021-03-18 - Initial report based on diff between commits 9bd0481...755d32e

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.