Quantstamp Security Assessment Certificate

February 6th 2023 — Quantstamp Verified

Cryptex Finance - Artbitrum Integration

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

Executive Summary

Type	Governance Bridge
Auditors	Ibrahim Abouzied, Auditing Engineer Guillermo Escobero, Security Auditor Sina Pilehchiha, Audit Engineer I Adrian Koegl, Security Engineer
Timeline	2022-12-05 through 2022-12-09
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Bridging Architecture Bridging Overview and Questions
Documentation Quality	High
Test Quality	Medium

Source Code

Repository	Commit
cruptexfinance/contracts	f4b2670 initial audit
cruptexfinance/contracts	5336501 fixes

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

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
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

Cryptex provides financial tools, such as their Total Market Cap Token, to help investors capture exposure to a broad range of cryptocurrencies and tokens. The Cryptex protocol is governed and upgraded by CTX token holders.
The scope of this audit is on the new contracts created to enact governance proposals cross-chain, with proposals starting from the ETH mainnet and bridging into Arbitrum. Other Cryptex technology and the TCAP token are outside of the scope of this audit. Users should consult the other audits performed on Cryptex for a full overview.
The main issues found are surrounding the intricate upgrade process for the contracts (QSP-3). This is further complicated by the deviations in contract ownership (QSP-1) and elevated permissions of the contract owner in `L1MessageRelayer` (QSP-2).

ID	Description	Severity	Status
QSP-1	Drift in Contract Ownership	^ Medium	Fixed
QSP-2	Owner of `L1MessageRelayer` Is Single Point of Failure	^ Medium	Fixed
QSP-3	Changes to `L2MessageExecutor` Are Error-Prone	^ Medium	Fixed
QSP-4	Missing Input Validation	⌄ Low	Fixed
QSP-5	Ownership Can Be Renounced	⌄ Low	Fixed
QSP-6	L1 to L2 Messages May Fail and Require Further Action	○ Informational	Acknowledged
QSP-7	Clone-and-Own	○ Informational	Acknowledged
QSP-8	Messages Cannot Pass Any Assets to L2	? 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 audit was performed on the following files only:

- `contracts/arbitrum/*`

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

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

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Findings

QSP-1 Drift in Contract Ownership

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `ArbitrumTreasury.sol`, `ArbitrumOrchestrator.sol`

Description: The `ArbitrumTreasury` and `ArbitrumOrchestrator` both override `onlyOwner` to check the `msg.sender` against the `arbitrumMessageExecutor` rather than the inherited `owner` variable. This means that ownership is changed through by calling `updateMessageExecutor()`. However, these contracts inherit the functions `transferOwnership()` and `renounceOwnership()` from the `Proprietor` contract. A user may call these functions incorrectly thinking they have changed the owner.

Recommendation: Consolidate these contracts to have one source of truth for the owner. Either disable `transferOwnership()` or override it to be a wrapper for `updateMessageExecutor()`. Disable `rencounceOwnership()`.

Update: The Cryptex team fixed the issue in commit `3920a51`. `updateMessageExecutor()` was removed and the ownership of the contract is now tracked the `Proprietor` contract.

QSP-2 Owner of `L1MessageRelayer` Is Single Point of Failure

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `L1MessageRelayer.sol`

Description: The access control to change the `L2MessageExecutor` in the `L1MessageRelayer` contract is centralized. In contrast, changing the `L1MessageRelayer` in the `L2MessageExecutor` contract requires a passed governance proposal. This means that the owner of the `L1MessageRelayer` contract has the ability to change the `L2MessageExecutor` without any checks or safeguards. This creates a single point of failure, as a malicious or compromised owner of the `L1MessageRelayer` contract could potentially render all contracts deployed on Arbitrum inaccessible and unchangeable.

Recommendation: To address this issue, we recommend decentralizing the access control to change the `L2MessageExecutor` in the `L1MessageRelayer` contract. This could be done by requiring a passed governance proposal in accordance with the design of changing the `L1MessageRelayer` address in `L2MessageExecutor`. This would ensure that updates to the `L2MessageExecutor` are done in a more secure and decentralized way, and remove the single point of failure.
It should be noted that this could give rise to another issue unless the recommendation in QSP-3, "Changes to `L2MessageExecutor` are error-prone", is followed. Furthermore, to allow for initializing the `L2MessageExecutor` address, it should be considered to allow an owner to initialize the address once.

Update: The Cryptex fixed the issue in commit `ab92123`. `updateL2MessageExecutor()` can only be called by the `timeLock`. A `setL2MessageExecutor()` function was added and configured to only be callable once.

QSP-3 Changes to `L2MessageExecutor` Are Error-Prone

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `L2MessageExecutor.sol`

Description: Updating the `L2MessageExecutor` contract is cumbersome and error-prone. When re-deploying the `L2MessageExecutor` contract, its address must be updated in the `L1MessageRelayer`, `Arbitrum Treasury`, and `Arbitrum Orchestrator` contracts. This process has two potential issues:

1. If a wrong address is set in `Arbitrum Treasury` or `Arbitrum Orchestrator`, they would be rendered unusable.
2. If the recommendation in "Owner of `L1MessageRelayer` is single point of failure" is followed, it should be ensured that the addresses in `Arbitrum Treasury` and `Arbitrum Orchestrator` are updated before `L1MessageRelayer`. If this is not done, the `L1MessageRelayer` contract will point to the new `L2MessageExecutor` address, and the `Arbitrum Treasury` and `Arbitrum Orchestrator` contracts will be inaccessible until a passed proposal updates the `L1MessageRelayer` contract to point to the old `L2MessageExecutor` address.

Recommendation: The `L2MessageExecutor` contract is the most interdependent in the current structure. To simplify the process of updating the `L2MessageExecutor` contract and reduce the risk of errors, we recommend implementing the Upgradeability pattern in the `L2MessageExecutor` contract. This pattern allows for logic upgrades without requiring updates to the pointers in the other contracts. This would improve the maintainability and reliability of the system.

Update: The Cryptex team fixed the issue in commit `101652b`. The `L2MessageExecutor` contract was made upgradeable and an `L2MessageExecutorProxy` contract was added.

QSP-4 Missing Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `L1MessageRelayer.sol`, `L2MessageExecutor.sol`

Description: Some functions do not validate their inputs, which can result in unexpected behavior by the contracts. A non-exhaustive list includes:

- `L1MessageRelayer.constructor()`
  - Validate that `_timeLock` and `_inbox` are not the zero address. If the `L1MessageRelayer` is deployed after the `L2MessageExecutor`, validate that `_l2MessageExecutor` is not the zero address.
- `L1MessageRelayer.relayMessage()`
  - Validate that `maxGas` and `gasPriceBid` are not one. Based on Arbitrum source code comments, if any of those parameters are one, the ticket creation will raise a `RetryableData` error.
- `L2MessageExecutor.executeMessage()`
  - In the statement `(bool success, ) = target.call(callData);` (Line #54), the `target` address does not have a zero address validation check.
- `L2MessageExecutor.constructor()`
  - If the `L2MessageExecutor` is deployed after the `L1MessageRelayer`, validate that `_l1MessageRelayer` is not the zero address.

Recommendation: Add the missing input validation.

Update: The Cryptex team fixed the issue in commit `069c9db`. The missing input validation was added.

QSP-5 Ownership Can Be Renounced

Severity: *Low Risk*

Status: Fixed

File(s) affected: `L1MessageRelayer.sol`

Description: It is possible that all contracts inheriting from `Ownable` are left without an owner calling `Ownable.renounceOwnership()`. All the functions modified by `onlyOwner` will be blocked.

Recommendation: If this is not expected, consider overriding `Ownable.renounceOwnership()` so that ownership cannot be renounced.

Update: The Cryptex team fixed the issue in commit `ab92123`. `L1MessageRelayer` is no longer `Ownable`.

QSP-6 L1 to L2 Messages May Fail and Require Further Action

Severity: *Informational*

Status: Acknowledged

File(s) affected: `L1MessageRelayer.sol`, `L2MessageExecutor.sol`

Description: When creating a Retryable Ticket in `L1MessageRelayer`, it is not guaranteed that the transaction will succeed. If the message fails to be redeemed, manual interaction is required to retry. The awareness of this matter is particularly important as redeemables may expire.

Recommendation: Make sure to implement a reliable mechanism to redeem the ticket should it initially fail. More details on the required manual interaction can be found here.

Update: The Cryptex team acknowledged the issue with the following message: "We will be developing a tool to monitor tickets so that we take action immediately in order to redeem the ticket."

QSP-7 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `AddressAliasHelper.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 a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve traceability of the file.

Update: The Cryptex team acknowledged the issue with the following message: "We are on solidity 0.7.5 and the Arbitrum's library is written for solidity > 0.8. We are maintaining our own library for compatibility reasons."

QSP-8 Messages Cannot Pass Any Assets to L2

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `L1MessageRelayer.sol`, `L2MessageExecutor.sol`

Description: Currently, `Inbox.createRetryableTicket()` is not used to transfer ETH from L1 to L2. If such functionality is desired in any potential governance decision, it cannot be achieved with the current implementation. This might be especially desirable if `executeMessage()` should be able to call a `payable` function on Arbitrum that requires a value to be passed on.

Recommendation: Consider whether any governance decisions could entail transferring assets from L1 to Arbitrum. If this is the case, make use of the `Inbox.depositEth()` function.

Update: The Cryptex team acknowledged the issue with the following message: "We have discussed internally and we do not see the need to transfer ETH from L1 to L2 via the bridge."

Adherence to Specification

The specification states `L1MessageRelayer can only be called by our Timelock contract.`. However, `L1MessageRelayer.updateL2MessageExecutor()` can only be called by the owner of the contract (not necessarily the `Timelock` contract).

Code Documentation

The current documentation provides a general overview of how the system works, but it does not include detailed information about initialization and maintenance. In particular, it is important to provide clear instructions on the intended order of deployment and how upgrades on contracts should be executed. This is especially important given the high level of interdependency between the different contracts in the system.

In addition to providing detailed instructions on how to initialize and maintain the system, it would also be helpful to include examples or case studies that demonstrate how the contracts can be used in practice. This would help users to better understand the intended usage of the contracts and how they fit into the overall system. By providing more comprehensive and specific documentation, the system can be made more accessible and easier to use.

Adherence to Best Practices

1. Rename `L2MessageExecutor.updateL2MessageRelayer()` to `updateL1MessageRelayer()`.
2. It is important to make error messages as specific as possible, but this can come at the cost of increased deployment gas costs. In the present contracts, it may be

worthwhile to consider removing the substrings that specify the function from which the error originates, without sacrificing specificity. This would help to reduce gas costs without reducing the usefulness of the error messages.

3. Contracts starting with "I", such as `ITreasury` or `IOrchestrator` usually indicate an interface. Therefore, this might be misleading when used in actual contract implementations. Consider renaming them.
4. Change the constant `AddressAliasHelper.offset` to the UPPER_CASE_WITH_UNDERSCORES format.
5. Some code statements do not have any effect on the execution and seem to be a mistake done while copying: After emitting `TransactionExecuted` event in `IOrchestrator`, `ITreasury` and `Orchestrator`

```
emit TransactionExecuted(target, value, signature, data);
(target, value, signature, data); // This line has no effect.
```

Test Results

Test Suite Results

The test suite was run with the command `yarn test` and `yarn ftest`.

Using smart contracts for testing is generally a good idea in this context. However, the current tests only cover simple interoperability checks and do not include cases with increased interdependencies.

We note that `test/arbitrum/ArbitrumTreasury.sol` should likely be renamed to `test/arbitrum/ArbitrumTreasury.t.sol`.

```
yarn run v1.22.15
$ npx hardhat test


  Chainlink Oracle
     ✓ ...should deploy the contract (75ms)
     ✓ ...should set the parameters
     ✓ ...should get the oracle answer

  ERC20 Vault
     ✓ ...should deploy the contract (247ms)
     ✓ ...should allow the owner to set the treasury address
     ✓ ...should return the token price
     ✓ ...should allow users to create a vault
     ✓ ...should get vault by id
     ✓ ...should allow user to stake collateral (78ms)
     ✓ ...should allow user to retrieve unused collateral (47ms)
     ✓ ...should return the correct minimal collateral required
     ✓ ...shouldn't allow minting above cap (68ms)
     ✓ ...should allow user to mint tokens (58ms)
     ✓ ...should allow token transfers
     ✓ ...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
     ✓ ...should calculate the burn fee
     ✓ ...should allow users to burn tokens (60ms)
     ✓ ...should update the collateral ratio
     ✓ ...should allow users to retrieve stake when debt is paid
     ✓ ...should test liquidation requirements (69ms)
     ✓ ...should get the required collateral for liquidation
     ✓ ...should get the liquidation reward
     ✓ ...should allow liquidators to return profits
     ✓ ...should allow users to liquidate users on vault ratio less than ratio (146ms)
     ✓ ...should allow owner to pause contract
     ✓ ...shouldn't allow contract calls if contract is paused
     ✓ ...should allow owner to unpause contract

  ETH Vault
     ✓ ...should deploy the contract (181ms)
     ✓ ...should allow the owner to set the treasury address
     ✓ ...should return the token price
     ✓ ...should allow users to create a vault
     ✓ ...should get vault by id
     ✓ ...should allow user to stake weth collateral (93ms)
     ✓ ...should allow user to stake eth collateral (40ms)
     ✓ ...should allow user to retrieve unused collateral on eth (49ms)
     ✓ ...should allow user to retrieve unused collateral on weth (46ms)
     ✓ ...should return the correct minimal collateral required
     ✓ ...should allow user to mint tokens (68ms)
     ✓ ...should allow users to get collateral ratio
     ✓ ...shouln't allow users to retrieve stake unless debt is paid
     ✓ ...should calculate the burn fee
     ✓ ...should allow users to burn tokens (51ms)
     ✓ ...should update the collateral ratio
     ✓ ...should allow users to retrieve stake when debt is paid
     ✓ ...should test liquidation requirements (61ms)
     ✓ ...should get the required collateral for liquidation
     ✓ ...should get the liquidation reward
     ✓ ...should allow liquidators to return profits
     ✓ ...should allow users to liquidate users on vault ratio less than ratio (128ms)
     ✓ ...should allow owner to pause contract
     ✓ ...shouldn't allow contract calls if contract is paused
     ✓ ...should allow owner to unpause contract

  Liquidity Mining Reward
     ✓ ...should deploy the contract (65ms)
     ✓ ...should set the constructor values
     ✓ ...should allow an user to stake
     ✓ ...should allow owner to fund the reward handler
     ✓ ...should allow user to earn rewards
     ✓ ...should allow user to retrieve rewards
     ✓ ...should allow user to withdraw
     ✓ ...should allow vault to exit
     ✓ ...shouldn't allow to earn after period finish
     ✓ ...should allow to claim vesting after vesting time

  MATIC Vault
     ✓ ...should deploy the contract (162ms)
     ✓ ...should allow the owner to set the treasury address
     ✓ ...should return the token price
     ✓ ...should allow users to create a vault
     ✓ ...should get vault by id
     ✓ ...should allow user to stake wmatic collateral (68ms)
     ✓ ...should allow user to stake eth collateral (40ms)
     ✓ ...should allow user to retrieve unused collateral on eth (45ms)
     ✓ ...should allow user to retrieve unused collateral on wmatic (58ms)
     ✓ ...should return the correct minimal collateral required
     ✓ ...should allow user to mint tokens (79ms)
     ✓ ...should allow users to get collateral ratio
     ✓ ...shouln't allow users to retrieve stake unless debt is paid
     ✓ ...should calculate the burn fee
     ✓ ...should allow users to burn tokens (52ms)
     ✓ ...should update the collateral ratio
     ✓ ...should allow users to retrieve stake when debt is paid
     ✓ ...should test liquidation requirements (61ms)
     ✓ ...should get the required collateral for liquidation
     ✓ ...should get the liquidation reward
     ✓ ...should allow liquidators to return profits
     ✓ ...should allow users to liquidate users on vault ratio less than ratio (126ms)
     ✓ ...should allow owner to pause contract
     ✓ ...shouldn't allow contract calls if contract is paused
     ✓ ...should allow owner to unpause contract

  Orchestrator Contract
     ✓ ...should deploy the contract (175ms)
     ✓ ...should set the owner
     ✓ ...should set the guardian
     ✓ ...should set vault ratio
     ✓ ...should set vault burn fee
     ✓ ...should set vault liquidation penalty
     ✓ ...should prevent liquidation penalty + 100 to be above ratio
     ✓ ...should pause the Vault (41ms)
```

```
    ✓ ...should unpause the vault
    ✓ ...should set the liquidation penalty to 0 on emergency (42ms)
    ✓ ...should set the burn fee to 0 on emergency (41ms)
    ✓ ...should be able to send funds to owner of orchestrator
    ✓ ...should enable the TCAP cap
    ✓ ...should set the TCAP cap
    ✓ ...should add vault to TCAP token
    ✓ ...should remove vault to TCAP token
    ✓ ...should allow to execute a custom transaction

  Reward Handler
    ✓ ...should deploy the contract (58ms)
    ✓ ...should set the constructor values
    ✓ ...should allow a vault to stake for a user
    ✓ ...should allow owner to fund the reward handler
    ✓ ...should allow user to earn rewards
    ✓ ...should allow user to retrieve rewards
    ✓ ...should allow vault to withdraw
    ✓ ...should allow vault to exit
    ✓ ...shouldn't allow to earn after period finish

  TCAP Token
    ✓ ...should deploy the contract (38ms)
    ✓ ...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

  ERC20 Vaults With Non 18 Decimal
    ✓ ...check collateralDecimalsAdjustmentFactor
    ✓ ...should have same amount of collateral in USD
    ✓ ...should have same Vault Ratio (91ms)
    ✓ ...should have same vault ratio after burning TCAP (135ms)
    ✓ ...should have same vault ratio after removing collateral (130ms)
    ✓ ...should have same vault ratio when vault ratio goes down (104ms)
    ✓ ...should have same requiredLiquidationTCAP when vault ratio goes down (122ms)
    ✓ ...should have same liquidationReward when vault ratio goes down (125ms)
    ✓ ...should have same vault ratio after liquidating vault (193ms)
    ✓ ...should be able to liquidate when vault ratio falls below 100 (90ms)
    ✓ ...should be able to burn TCAP when vault ratio falls below 100 (77ms)

  Ctx
    ✓ ...should permit (56ms)
    ✓ ...should changes allowance (40ms)
    ✓ ...should allow nested delegation (74ms)
    ✓ ...should mint (57ms)

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

  scenario:TreasuryVester
    ✓ setRecipient:fail
    ✓ claim:fail
    ✓ claim:~half (43ms)
    ✓ claim:all (49ms)

  Polygon Integration Test
    ✓ ...Add new vault without Governance
    ✓ ...Transfer OwnerShip to DAO post setup (39ms)
    ✓ ...Add new vault through Governance (5402ms)

  PolygonL2Messenger Test
    ✓ ...Successful Message Execution
    ✓ ...Do not allow non owner to execute Message
    ✓ ... revert for unauthorized Fxchild
    ✓ ... revert for unauthorized direct call to PolygonMsgTester


  150 passing (19s)

 ✓  Done in 48.99s.

FORGE RESULTS:
Running 3 tests for test/arbitrum/ArbitrumOrchestartor.t.sol:ArbitrumOrchestratorTest
[PASS] testNewOwnerCanMakeCalls() (gas: 494319)
[PASS] testRenounceOwnershipShouldRevert() (gas: 477062)
[PASS] testUpdateOwner() (gas: 480808)
Test result: ok. 3 passed; 0 failed; finished in 3.86ms

Running 4 tests for test/LinkAave.t.sol:LinkAaveTest
[PASS] testBurnTCAP() (gas: 167)
[PASS] testDepositCollateral() (gas: 189)
[PASS] testMintTCAP() (gas: 166)
[PASS] testRemoveCollateral() (gas: 144)
Test result: ok. 4 passed; 0 failed; finished in 4.05ms

Running 2 tests for test/arbitrum/ArbitrumTreasury.sol:ArbitrumTreasuryTest
[PASS] testRenounceOwnershipShouldRevert() (gas: 477033)
[PASS] testUpdateOwner() (gas: 480651)
Test result: ok. 2 passed; 0 failed; finished in 4.00ms

Running 1 test for test/arbitrum/GovernanceBridgeIntegration.t.sol:GovernanceBridgeIntegration
[PASS] testAddVault() (gas: 904481)
Test result: ok. 1 passed; 0 failed; finished in 6.63ms

Running 5 tests for test/OptimisticTreasury.t.sol:OptimisticTreasuryTest
[PASS] testExecuteTransaction() (gas: 1002277)
[PASS] testRenounceOwnership() (gas: 17826)
[PASS] testRetrieveEth() (gas: 50194)
[PASS] testSetParams() (gas: 10604)
[PASS] testTransferOwnership(address) (runs: 256, μ: 23001, ~: 23021)
Test result: ok. 5 passed; 0 failed; finished in 15.09ms

Running 19 tests for test/VaultsPausing.t.sol:VaultDisablingTest
[PASS] testAddCollateralETH_ShouldRevert_WhenIsDisabled() (gas: 139434)
[PASS] testAddCollateralETH_ShouldWork_WhenToogleDisabledFalse() (gas: 190020)
[PASS] testAddCollateral_ShouldRevert_WhenIsDisabled() (gas: 193321)
[PASS] testAddCollateral_ShouldWork_WhenToogleDisabledFalse() (gas: 198077)
[PASS] testBurn_ShouldNotBurn_WhenIsDisabled() (gas: 327895)
[PASS] testBurn_ShouldWork_WhenToogleDisabledFalse() (gas: 315196)
[PASS] testCreateVault_ShouldRevert_WhenIsDisabled() (gas: 46080)
[PASS] testCreateVault_ShouldWork_WhenToogleDisabledFalse() (gas: 110546)
[PASS] testLiquidateVault_ShouldNotLiquidate_WhenIsDisabled() (gas: 331092)
[PASS] testLiquidateVault_ShouldWork_WhenToogleDisabledFalse() (gas: 387697)
[PASS] testMint_ShouldRevert_WhenIsDisabled() (gas: 198735)
[PASS] testMint_ShouldWork_WhenToogleDisabledFalse() (gas: 296331)
[PASS] testRemoveCollateralETH_ShouldRevert_WhenIsDisabled() (gas: 198693)
[PASS] testRemoveCollateralETH_ShouldWork_WhenToogleDisabledFalse() (gas: 202766)
[PASS] testRemoveCollateral_ShouldRevert_WhenIsDisabled() (gas: 198716)
[PASS] testRemoveCollateral_ShouldWork_WhenToogleDisabledFalse() (gas: 187287)
[PASS] testToggleFunction_ShouldDisableFunction() (gas: 40157)
[PASS] testToggleFunction_ShouldOnlyDisableOneFunction_WhenToogled() (gas: 51430)
[PASS] testToggleFunction_ShouldRevert_WhenNotOwner() (gas: 13872)
Test result: ok. 19 passed; 0 failed; finished in 64.74ms

Running 13 tests for test/arbitrum/ArbitrumMessages.t.sol:ArbitrumMessages
[PASS] testExecuteMessage() (gas: 36013)
[PASS] testExecuteMessageThroughL1Relayer() (gas: 58088)
[PASS] testL1MessageRelayerRenounceOwnership() (gas: 13069)
[PASS] testL2MessageExecutorInializedOnlyOnce() (gas: 457476)
[PASS] testRevertForZeroInboxAddress() (gas: 62225)
[PASS] testRevertForZeroL1MessageRelayerAddress() (gas: 452331)
[PASS] testRevertForZeroTimelockAddress() (gas: 62151)
[PASS] testRevertOnUnAuthorizedTimelock() (gas: 14519)
[PASS] testRevertOnUpdateExecutor() (gas: 12672)
[PASS] testRevertWhenZeroTargetAddress() (gas: 20278)
[PASS] testRevertsetL2MessageExecutorProxyAlreadySet() (gas: 15423)
[PASS] testRevertsetL2MessageExecutorProxyCalledByNotOwner() (gas: 12745)
[PASS] testUpdateL2MessageExecutor() (gas: 22195)
Test result: ok. 13 passed; 0 failed; finished in 118.02ms

Running 17 tests for test/HardETHVaultHandler.t.sol:ETHVaultHandlerTest
[PASS] testBurnTCAP_ShouldBurn_WhenFeeIsPaid(uint96) (runs: 256, μ: 86228, ~: 327)
[PASS] testBurnTCAP_ShouldRevert_WhenFeeIsNotPaid(uint96) (runs: 256, μ: 99960, ~: 306)
[PASS] testConstructor_ShouldRevert_WhenBurnFeeIsHigh(uint256) (runs: 256, μ: 1147112, ~: 110973)
[PASS] testConstructor_ShouldRevert_WhenLiquidationPenaltyIsHigh(uint256,uint256) (runs: 256, μ: 1995436, ~: 108918)
[PASS] testConstructor_ShouldSetParams_WhenInitialized() (gas: 51229)
[PASS] testGetFee_ShouldCalculateCorrectValue(uint8,uint96) (runs: 256, μ: 58770, ~: 59346)
[PASS] testGetFee_ShouldCalculateCorrectValue_withNewDecimalFormat(uint8,uint96) (runs: 256, μ: 58620, ~: 59298)
[PASS] testLiquidateVault_ShouldLiquidateVault_WhenRatioAbove100(uint96) (runs: 256, μ: 468474, ~: 493112)
[PASS] testLiquidateVault_ShouldLiquidateVault_WhenRatioBelow100(uint96) (runs: 256, μ: 476259, ~: 490969)
[PASS] testMint_ShouldCreateTCAP() (gas: 278166)
[PASS] testMint_ShouldMint_WhenEnoughTCAP() (gas: 424105)
[PASS] testSetBurnFee_ShouldAllowDecimals_WhenValueBelowMax(uint256) (runs: 256, μ: 6077, ~: 315)
[PASS] testSetBurnFee_ShouldRevert_WhenNotCalledByOwner() (gas: 14575)
[PASS] testSetBurnFee_ShouldRevert_WhenValueAboveMax(uint256) (runs: 256, μ: 20642, ~: 26335)
[PASS] testSetLiquidationPenalty_ShouldUpdateValue(uint256) (runs: 256, μ: 22383, ~: 26723)
```

```
[PASS] testSetMinimumTCAP_ShouldUpdateValue(uint256) (runs: 256, μ: 36221, ~: 36353)
[PASS] testShouldUpdateRatio(uint256) (runs: 256, μ: 27666, ~: 29000)
Test result: ok. 17 passed; 0 failed; finished in 117.94ms
```

Code Coverage

Test coverage was gathered by running `yarn coverage` and `forge coverage` respectively. The test show high coverage, though `L1MessageRelayer` has room for improvement for covering more branches.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	81.47	62.38	78.74	81.31	
BaseOrchestrator.sol	13.16	11.54	21.05	16.67	… 340,344,345
BaseTreasury.sol	0	0	33.33	0	… 63,72,73,74
ERC20VaultHandler.sol	100	100	100	100	
ETHVaultHandler.sol	93.33	70	100	93.75	118
IVaultHandler.sol	94.62	79.03	90	94.74	… 616,620,621
IWETH.sol	100	100	100	100	
IWMATIC.sol	100	100	100	100	
LiquidityReward.sol	87.3	68.18	88.24	87.5	… 246,319,320
MATICVaultHandler.sol	93.33	70	100	93.75	115
Orchestrator.sol	94.74	76.92	94.74	92.86	96,103,312
Proprietor.sol	66.67	33.33	50	60	41,42,53,54
RewardHandler.sol	89.29	66.67	88.89	89.66	… 236,240,241
TCAP.sol	100	87.5	100	100	
contracts/arbitrum/	0	0	0	0	
AddressAliasHelper.sol	0	100	0	0	33,45
ArbitrumOrchestrator.sol	0	100	0	0	24
ArbitrumTreasury.sol	0	100	0	0	21
L1MessageRelayer.sol	0	0	0	0	… 76,80,90,91
L2MessageExecutor.sol	0	0	0	0	… 51,55,56,57
L2MessageExecutorProxy.sol	100	100	0	100	
contracts/governance/	77.03	49.43	77.59	76.37	
Ctx.sol	85.61	56.76	92	85.61	… 522,524,624
GovernorBeta.sol	69.79	44.64	71.43	67.39	… 455,456,459
Timelock.sol	57.14	34.38	44.44	57.14	… 166,168,202
TreasuryVester.sol	94.74	66.67	100	94.74	57
contracts/optimism/	0	0	0	0	
OptimisticOrchestrator.sol	0	0	0	0	27,31,39,44
OptimisticTreasury.sol	0	0	0	0	24,28,36,41
iOVM_CrossDomainMessenger.sol	100	100	100	100	
iOVM_L2CrossDomainMessenger.sol	100	100	100	100	
contracts/polygon/	76.67	50	76.92	76.47	
PolygonL2Messenger.sol	83.33	64.29	85.71	85	118,122,123
PolygonOrchestrator.sol	100	50	100	100	
PolygonTreasury.sol	33.33	16.67	33.33	28.57	38,43,54,58,62
All files	**75.76**	**51.61**	**71.43**	**75.16**	

File	% Lines	% Statements	% Branches	% Funcs
contracts/ BaseOrchestrator.sol	9.38% (3/32)	9.09% (3/33)	6.25% (1/16)	7.14% (1/14)
contracts/ BaseTreasury.sol	91.67% (11/12)	92.31% (12/13)	62.50% (5/8)	100.00% (2/2)
contracts/ ETHVaultHandler.sol	**93.33% (14/15)**	**93.75% (15/16)**	**30.00% (3/10)**	**100.00% (2/2)**
contracts/ IVaultHandler.sol	89.72% (96/107)	91.60% (120/131)	54.17% (26/48)	80.77% (21/26)

File	% Lines	% Statements	% Branches	% Funcs
contracts/ LiquidityReward.sol	0.00% (0/52)	0.00% (0/57)	0.00% (0/20)	0.00% (0/15)
contracts/ MATICVaultHandler.sol	0.00% (0/15)	0.00% (0/16)	0.00% (0/10)	0.00% (0/2)
contracts/ Orchestrator.sol	37.50% (12/32)	39.39% (13/33)	18.75% (3/16)	35.71% (5/14)
contracts/ Proprietor.sol	100.00% (5/5)	100.00% (5/5)	50.00% (1/2)	100.00% (2/2)
contracts/ RewardHandler.sol	0.00% (0/45)	0.00% (0/48)	0.00% (0/18)	0.00% (0/15)
contracts/ TCAP.sol	53.33% (8/15)	53.33% (8/15)	16.67% (1/6)	62.50% (5/8)
contracts/arbitrum/ AddressAliasHelper.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
contracts/arbitrum/ ArbitrumOrchestrator.sol	100.00% (1/1)	100.00% (1/1)	100.00% (0/0)	100.00% (1/1)
contracts/arbitrum/ ArbitrumTreasury.sol	100.00% (1/1)	100.00% (1/1)	100.00% (0/0)	100.00% (1/1)
contracts/arbitrum/ L1MessageRelayer.sol	90.91% (10/11)	92.31% (12/13)	50.00% (4/8)	100.00% (4/4)
contracts/arbitrum/ L2MessageExecutor.sol	100.00% (9/9)	100.00% (11/11)	90.00% (9/10)	100.00% (2/2)
contracts/governance/ Ctx.sol	3.73% (5/134)	3.25% (5/154)	4.17% (3/72)	4.17% (1/24)
contracts/governance/ GovernorBeta.sol	67.37% (64/95)	64.55% (71/110)	41.07% (23/56)	55.00% (11/20)
contracts/governance/ Timelock.sol	52.63% (20/38)	54.76% (23/42)	28.57% (8/28)	37.50% (3/8)
contracts/governance/ TreasuryVester.sol	0.00% (0/9)	0.00% (0/9)	0.00% (0/6)	0.00% (0/2)
contracts/mocks/ AAVE.sol	50.00% (1/2)	50.00% (1/2)	100.00% (0/0)	50.00% (1/2)
contracts/mocks/ AggregatorInterface.sol	50.00% (1/2)	50.00% (1/2)	100.00% (0/0)	50.00% (1/2)
contracts/mocks/ AggregatorInterfaceStable.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
contracts/mocks/ AggregatorInterfaceTCAP.sol	66.67% (2/3)	66.67% (2/3)	100.00% (0/0)	66.67% (2/3)
contracts/mocks/ CrossChainMsgTester.sol	0.00% (0/1)	0.00% (0/1)	100.00% (0/0)	0.00% (0/1)
contracts/mocks/ DAI.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
contracts/mocks/ FxRoot.sol	0.00% (0/4)	0.00% (0/5)	0.00% (0/2)	0.00% (0/2)
contracts/mocks/ GovernorAlpha.sol	0.00% (0/93)	0.00% (0/108)	0.00% (0/54)	0.00% (0/19)
contracts/mocks/ Greeter.sol	100.00% (2/2)	100.00% (2/2)	100.00% (0/0)	100.00% (2/2)
contracts/mocks/ LINK.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
contracts/mocks/ MockFxChild.sol	0.00% (0/5)	0.00% (0/6)	0.00% (0/2)	0.00% (0/2)
contracts/mocks/ StateSender.sol	0.00% (0/7)	0.00% (0/7)	0.00% (0/4)	0.00% (0/2)
contracts/mocks/ USDC.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
contracts/mocks/ WBTC.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
contracts/mocks/ WETH.sol	94.74% (18/19)	94.74% (18/19)	50.00% (4/8)	83.33% (5/6)
contracts/mocks/ WMATIC.sol	0.00% (0/19)	0.00% (0/19)	0.00% (0/8)	0.00% (0/6)
contracts/oracles/ ChainlinkOracle.sol	23.81% (5/21)	20.83% (5/24)	25.00% (2/8)	12.50% (1/8)
contracts/polygon/ PolygonL2Messenger.sol	0.00% (0/14)	0.00% (0/16)	0.00% (0/10)	0.00% (0/4)
contracts/polygon/ PolygonOrchestrator.sol	0.00% (0/3)	0.00% (0/3)	0.00% (0/2)	0.00% (0/1)
contracts/polygon/ PolygonTreasury.sol	0.00% (0/3)	0.00% (0/3)	0.00% (0/2)	0.00% (0/1)
test/OptimisticTreasury. t.sol	100.00% (4/4)	100.00% (4/4)	100.00% (0/0)	100.00% (4/4)
test/arbitrum/mocks/ MockInbox.sol	**100.00% (8/8)**	**100.00% (8/8)**	**50.00% (1/2)**	**100.00% (1/1)**
Total	35.29% (300/850)	35.82% (341/952)	21.56% (94/436)	32.50% (78/240)

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

```
95b737601c9798980d6a70abae2781354698ae1a1940eb6019b861a99cacee97 ./contracts/arbitrum/ArbitrumTreasury.sol
4bf21c0d573720b483b8903c14a5a3ffe750741ce21c0df76cf1c77d3298d535 ./contracts/arbitrum/L1MessageRelayer.sol
ae9a6bea21561148856cff29c1cfc35c0cb2abf42956f6c6dfdbd29a5944b709 ./contracts/arbitrum/AddressAliasHelper.sol
1006428352ecd0c3e1c00ce1e02652cdcc8d4b87de69622de8b8333bd1a2da38 ./contracts/arbitrum/ArbitrumOrchestrator.sol
f2d2f1c2b3e5c5c7a37a8b2df8ffb6b4705bd00df360a444687f25fb61c68060 ./contracts/arbitrum/L2MessageExecutor.sol
```

Tests

```
ce6159f82f9e4a9842e8acc8690b5b34b746ad54e1ce2240b6dda35d249021d1 ./test/arbitrum/ArbitrumMessages.t.sol
897a73b6ec1830c59086b4ab5d8a7b4afb05687decd47c83d0f76e061f772fa3 ./test/arbitrum/GovernanceBridgeIntegration.t.sol
2e8e60897304e0c1666dbbca3a95baf54662a695bace04c895e90e2b97bf1e7a ./test/arbitrum/mocks/MockInbox.sol
```

Changelog

- 2022-12-09 - Initial report
- 2022-12-21 - Fix-Review

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.