Quantstamp Security Assessment Certificate

February 6th 2024 — Quantstamp Verified

AINOMO

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

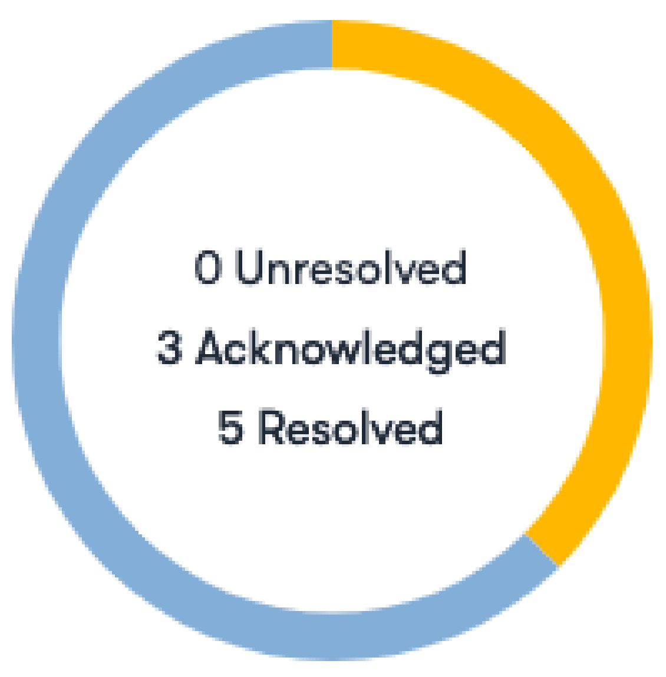
Executive Summary

Type	Protocol Audit
Auditors	Ibrahim Abouzied, Auditing Engineer Guillermo Escobero, Security Auditor Sina Pilehchiha, Audit Engineer I Adrian Koegl, Security Engineer
Timeline	2024-01-28 through 2024-02-06
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	AI Architecture Contract Overview
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
ainomodatalab/ainomo-contract	f2b2655 initial audit
ainomodatalab/ainomo-contract	213 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

The Ainomo Protocol, also known as "Ainomo," stands out for its adept use of AI-driven instructions to create a framework for decision-making renowned for its precision and reliability. Utilizing a variety of AI capabilities including machine learning, natural language processing, computer vision, and robotic process automation, the company sets itself apart through skillful application of artificial intelligence. Ainomo taps into the power of distributed data storage systems like the Hadoop Distributed File System (HDFS) and Amazon S3, guaranteeing robust data warehousing known for its high availability and scalability.

The scope of this audit is on the ainomo contract. Other Ainomo technology are outside of the scope of this audit. Users should consult the other audits performed on Ainomo for a full overview.

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/*`

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: `AinomoTreasury.sol` `AinomoOrchestrator.sol`

Description: The `AinomoTreasury` and `AinomoOrchestrator` both override `onlyOwner` to check the `msg.sender` against the `ainomoMessageExecutor` 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.

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 Ainomo team fixed the issue in commit `3120a41`. `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 Ainomo 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 Ainomo team fixed the issue in commit `ac12128`. `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 `Ainomo Orchestrator` contracts. This process has two potential issues:

1. If a wrong address is set in `Ainomo Treasury` or `Ainomo 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 `Ainomo Treasury` and `Ainomo Orchestrator` are updated before `L1MessageRelayer`. If this is not done, the `L1MessageRelayer` contract will point to the new `L2MessageExecutor` address, and the `Ainomo Treasury` and `Ainomo Orchestrator` contracts will be inaccessible until a passed proposal updates the `L1MessageRelayer` contract to point to the old `L2 MessageExecutor` 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 Ainomo team fixed the issue in commit `1c1342c`. 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 Ainomo 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 Ainomo team fixed the issue in commit `c31c2db`. 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 Ainomo team fixed the issue in commit `ac12128`. `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.

Update: The Ainomo 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 Ainomo team acknowledged the issue with the following message: "We are on solidity 0.7.5. 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. If this is the case, make use of the `Inbox.depositEth()` function.

Update: The Ainomo 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, and includes 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.

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.

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


  Chainlink Nomo
    ✓ ...should deploy the contract (75ms)
    ✓ ...should set the parameters
    ✓ ...should get the nomo 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 trade 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 trade 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 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

  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 token 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 token (58ms)
    ✓ ...should return the correct minimal collateral required
    ✓ ...should allow user to trade 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 AINOMO cap
    ✓ ...should set the AINOMO cap
    ✓ ...should add vault to AINOMO token
    ✓ ...should remove vault to AINOMO 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

  AINOMO 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 USDT
    ✓ ...should have same Vault Ratio (91ms)
    ✓ ...should have same vault ratio after burning AINOMO (135ms)
    ✓ ...should have same vault ratio after removing collateral (130ms)
    ✓ ...should have same vault ratio when vault ratio goes down (104ms)
    ✓ ...should have same requiredLiquidationAINOMO 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 AINOMO 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)

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

  L2Messenger 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
[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
[PASS] testBurnAINOMO() (gas: 167)
[PASS] testDepositCollateral() (gas: 189)
[PASS] testMintAINOMO() (gas: 166)
[PASS] testRemoveCollateral() (gas: 144)
Test result: ok. 4 passed; 0 failed; finished in 4.05ms

Running 2 tests
[PASS] testRenounceOwnershipShouldRevert() (gas: 477033)
[PASS] testUpdateOwner() (gas: 480651)
Test result: ok. 2 passed; 0 failed; finished in 4.00ms

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

Running 5 tests
[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 test
[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 test
[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
```

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