Quantstamp Security Assessment Certificate

February 3rd 2021 — Quantstamp Verified

Skale Proxy Contracts

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

Executive Summary

Type	DeFi
Auditors	Jake Goh Si Yuan, Senior Security Researcher Jan Gorzny, Blockchain Researcher Kevin Feng, Blockchain Researcher
Timeline	2020-11-16 through 2021-01-26
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Provided Documentation
Documentation Quality	High
Test Quality	Medium

Source Code

Repository	Commit
IMA/proxy	8ba7484
None	ee72736
None	082b932

Total Issues	5 (4 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	2 (1 Resolved)
Informational Risk Issues	2 (2 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

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

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

Summary of Findings

We have performed a complete assessment of the codebase provided and discovered 5 issues of varying severities, amongst which there is 1 high, 2 low and 2 informational. We urge the Skale team to address these issues and consider our recommendations with which to go about fixing it. Overall, we have found the codebase to be of good quality with well named methods and inline documentation. That being said, there are some room for improvement with regards to documentation consistency.
At the same time, it is important to note that we acknowledge that there exists a node.js agent that handles the communication between mainnet and the separate chains. As this audit was focused only on the smart contracts components, that part is out of scope of the audit and might be a source of centralization for attacks.

ID	Description	Severity	Status
QSP-1	Improper access control to a core method	High	Fixed
QSP-2	Integer Overflow / Underflow	Low	Fixed
QSP-3	Race Conditions / Front-Running	Low	Acknowledged
QSP-4	Schain ETH contract is supply limited	Informational	Fixed
QSP-5	Hardcoded addresses and associated methods with unknown results	Informational	Mitigated

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 v0.6.13

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 Improper access control to a core method

Severity: *High Risk*

Status: Fixed

File(s) affected: `OwnableForMainnet.sol`, `OwnableForSchain.sol`

Description: `OwnableForMainnet` is intended to be a basic singular access control inheritable contract that is used by `LockAndDataForMainnet`. The logic of this contract is very similar to a well known and ubiquitous Ownable implementation provided by OpenZeppelin, with a major difference in an inclusion of a method `setOwner`.
The `setOwner` method is used via `transferOwnership` to set the new `_ownerAddress`. However, this method is set to `public` visibility, which means that it can be executed by any arbitrary actor to any arbitrary value. This is extremely dangerous given the relative importance and power of the owner role.

Recommendation: Use OpenZeppelin's implementation instead, as it has already been done for many other contracts, instead of rolling a new owner-logic contract. Otherwise, ensure that `setOwner` is either set to `internal` or armed with some access control.

QSP-2 Integer Overflow / Underflow

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LockAndDataForMainnet.sol`, `LockAndDataForSchain.sol`,

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. Integer overflow and underflow may cause many unexpected kinds of behavior and was the core reason for the `batchOverflow` attack. Here's an example with `uint8` variables, meaning unsigned integers with a range of `0..255`. `function under_over_flow() public { uint8 num_players = 0; num_players = num_players - 1; // 0 - 1 now equals 255! if (num_players == 255) { emit LogUnderflow(); // underflow occurred } uint8 jackpot = 255; jackpot = jackpot + 1; // 255 + 1 now equals 0! if (jackpot == 0) { emit LogOverflow(); // overflow occurred } }`
We have discovered these instances in the codebase:

1. LockAndDataForMainnet.sol::L144 `approveTransfers[to] += amount;`
2. LockAndDataForSchain.sol::L206 `ethCosts[to] += amount`
3. MessageProxyForSchain.sol::L428 `_idxHead += cntDeleted`
4. MessageProxyForSchain.sol::L416 `++ _idxTail;`
5. MessageProxyForSchain.sol::L334 `connectedChains[keccak256(abi.encodePacked(srcChainID))].incomingMessageCounter += uint256(messages.length);`
6. MessageProxyForSchain.sol::L340 `connectedChains[keccak256(abi.encodePacked(schainName))].incomingMessageCounter++;`
7. MessageProxyForSchain.sol::L252 `connectedChains[dstChainHash].outgoingMessageCounter++;`
8. MessageProxyForMainnet.sol::L532 `_idxHead += cntDeleted`
9. MessageProxyForMainnet.sol::L517 `++ _idxTail;`
10. MessageProxyForMainnet.sol::L315`connectedChains[keccak256(abi.encodePacked(srcChainID))].incomingMessageCounter += uint256(messages.length);`
11. MessageProxyForMainnet.sol::L330 `connectedChains[keccak256(abi.encodePacked(schainName))].incomingMessageCounter++;`
12. MessageProxyForMainnet.sol::L247 `connectedChains[dstChainHash].outgoingMessageCounter++;`

Recommendation: Use SafeMath for all instances of arithmetic.

QSP-3 Race Conditions / Front-Running

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `EthERC20.sol`

Related Issue(s): SWC-114

Description: A block is an ordered collection of transactions from all around the network. It's possible for the ordering of these transactions to manipulate the end result of a block. A miner attacker can take advantage of this by generating and moving transactions in a way that benefits themselves.
In particular, this refers to the well known `approve` frontrunning attack on ERC20.

Exploit Scenario: Imagine two friends — Alice and Bob.

1. Alice decides to allow Bob to spend some of her funds, for example, 1000 tokens. She calls the approve function with the argument equal to 1000.
2. Alice rethinks her previous decision and now she wants to allow Bob to spend only 300 tokens. She calls the approve function again with the argument value equal to 300.
3. Bob notices the second transaction before it is actually mined. He quickly sends the transaction that calls the transferFrom function and spends 1000 tokens.
4. Since Bob is smart, he sets very high fee for his transaction, so that miner will definitely want to include his transaction in the block. If Bob is as quick as he is generous, his transaction will be executed before the Alice's one.
5. In that case, Bob has already spent 1000 Alice's tokens. The number of Alice's tokens that Bob can transfer is equal to zero. 6.Then the Alice's second transaction is mined. That means, that the Bob's allowance is set to 300. 7.Now Bob can spend 300 more tokens by calling the transferFrom function. As a result, Bob has spent 1300 tokens. Alice has lost 1000 tokens and one friend.

Recommendation: Make this issue well known such that users who use the allowance feature would be aware of it in such transitions. One may also include some defensive programming by allowing changes to only go to 0 or from 0.

QSP-4 Schain ETH contract is supply limited

Severity: *Informational*

Status: Fixed

File(s) affected: `EthERC20.sol`

Description: The ETH contract on Schain that acts as an analogue token for the native ETH token on mainnet is limited by a variable `_capacity` that cannot be increased beyond an initially declared `120 * (10 ** 6) * (10 ** 18)`. However, given that the supply of ETH is not hard limited, it means that this contract would not be able to mint beyond that value.

Recommendation: Have some methods to change `_capacity`.

QSP-5 Hardcoded addresses and associated methods with unknown results

Severity: *Informational*

Status: Mitigated

File(s) affected: `LockAndDataForSchain.sol`, `MessageProxyForSchain.sol`, `LockAndDataOwnable.sol`, `OwnableForSchain.sol`, `PermissionsForSchain.sol`, `TokenManager.sol`

Description: There are some hardcoded addresses within the predeployed section, used within some of the key methods of the some of the contracts. As we are not able to see the logic that is predeployed and its' exact effects, we will not be able to certify methods utilizing these logic:

1. In LockAndDataForSchain.sol, the method `_checkPermitted`.
2. In LockAndDataForSchain.sol, the method `getEthERC20Address`.
3. In LockAndDataOwnable.sol, the method `getOwner`.
4. In MessageProxyForSchain.sol, the method `getChainID`.
5. In MessageProxyForSchain.sol, the method `getOwner`.
6. In MessageProxyForSchain.sol, the method `checkIsAuthorizedCaller`.
7. In OwnableForSchain.sol, the method `getOwner`.
8. In PermissionsForSchain.sol, the method `getLockAndDataAddress`.
9. In TokenManager.sol, the method `getChainID`.
10. In TokenManager.sol, the method `getProxyForSchainAddress`.

Update: The reaudit commit has refactored the approach but the issue remains the same that any logic approaching address `0xC033b369416c9Ecd8e4A07AaFA8b06b4107419E2` is opaque to the audit unless there is an independent way for the auditors to retrieve and check the data on `0x00c033b369416c9ecd8e4a07aafa8b06b4107419e2`.
Update: From the Skale team : "One note about QSP-5, the hard coded address 0xC033b369416c9Ecd8e4A07AaFA8b06b4107419E2 refers to the predeployed address for SkaleFeatures contract. Searching the repo for that address will show you the deployment scripts, that make SkaleFeatures accessible to the schain IMA system. I believe with this info, the issue is effectively resolved."
Update: Due to the zeal and information provided by the Skale team, we have decided to upgrade the status from `Unresolved` to `Mitigated`. It will remain the recommendation of the Quantstamp team that users independently verify that the hardcoded address has the expected contract code.

Automated Analyses

Slither

All of the results were checked through and were flagged as false positives. The following are best practices recommendations that should be adhered to :

```
getChainID() should be declared external:
    - MessageProxyForSchain.getChainID() (predeployed/MessageProxyForSchain.sol#349-357)
setOwner(address) should be declared external:
    - MessageProxyForSchain.setOwner(address) (predeployed/MessageProxyForSchain.sol#369-371)
verifyOutgoingMessageData(uint256,address,address,address,uint256) should be declared external:
    - MessageProxyForSchain.verifyOutgoingMessageData(uint256,address,address,address,uint256) (predeployed/MessageProxyForSchain.sol#386-401)
initialize(string,address) should be declared external:
    - MessageProxyForMainnet.initialize(string,address) (MessageProxyForMainnet.sol#398-403)
verifyOutgoingMessageData(uint256,address,address,address,uint256) should be declared external:
    - MessageProxyForMainnet.verifyOutgoingMessageData(uint256,address,address,address,uint256) (MessageProxyForMainnet.sol#408-423)
mint(address,uint256) should be declared external:
    - ERC20OnChain.mint(address,uint256) (predeployed/TokenFactory.sol#60-64)
getLockAndDataAddress() should be declared external:
    - PermissionsForMainnet.getLockAndDataAddress() (PermissionsForMainnet.sol#70-72)
logMessage(string) should be declared external:
    - SkaleFeatures.logMessage(string) (predeployed/SkaleFeatures.sol#60-62)
logDebug(string) should be declared external:
    - SkaleFeatures.logDebug(string) (predeployed/SkaleFeatures.sol#64-66)
logTrace(string) should be declared external:
    - SkaleFeatures.logTrace(string) (predeployed/SkaleFeatures.sol#68-70)
logWarning(string) should be declared external:
    - SkaleFeatures.logWarning(string) (predeployed/SkaleFeatures.sol#72-74)
logError(string) should be declared external:
    - SkaleFeatures.logError(string) (predeployed/SkaleFeatures.sol#76-78)
logFatal(string) should be declared external:
    - SkaleFeatures.logFatal(string) (predeployed/SkaleFeatures.sol#80-82)
getConfigVariableUint256(string) should be declared external:
    - SkaleFeatures.getConfigVariableUint256(string) (predeployed/SkaleFeatures.sol#84-97)
getConfigVariableAddress(string) should be declared external:
    - SkaleFeatures.getConfigVariableAddress(string) (predeployed/SkaleFeatures.sol#99-112)
getConfigVariableString(string) should be declared external:
    - SkaleFeatures.getConfigVariableString(string) (predeployed/SkaleFeatures.sol#114-126)
concatenateStrings(string,string) should be declared external:
    - SkaleFeatures.concatenateStrings(string,string) (predeployed/SkaleFeatures.sol#128-148)
getConfigPermissionFlag(address,string) should be declared external:
    - SkaleFeatures.getConfigPermissionFlag(address,string) (predeployed/SkaleFeatures.sol#150-165)
name() should be declared external:
    - EthERC20.name() (predeployed/EthERC20.sol#84-86)
symbol() should be declared external:
    - EthERC20.symbol() (predeployed/EthERC20.sol#92-94)
decimals() should be declared external:
    - EthERC20.decimals() (predeployed/EthERC20.sol#109-111)
increaseAllowance(address,uint256) should be declared external:
    - EthERC20.increaseAllowance(address,uint256) (predeployed/EthERC20.sol#194-197)
decreaseAllowance(address,uint256) should be declared external:
    - EthERC20.decreaseAllowance(address,uint256) (predeployed/EthERC20.sol#213-220)
```

Code Documentation

1. [FIXED] In EthERC20.sol::L48 to be consistent with other type declarations, `uint` -> `uint256` .
2. In LockAndDataForSchain.sol::L234 `sendEth` should be `sendETH`.
3. In LockAndDataForSchain.sol::L242 `receiveEth` should be `receiveETH`.

4. In LockAndDataForSchain.sol::L250 `getEthERC20Address` should be `getETH_ERC20Address`.
5. [FIXED] In LockAndDataForSchain.sol::L260 `name and adress are permitted` -> `name and address are permitted`.
6. In TokenManager.sol::[L528,L521] `addEthCost` -> `addETHCost`.
7. In TokenManager.sol::L119 `addEthCostWithoutAddress` -> `addETHCostWithoutAddress`.
8. [FIXED] In MessageProxyForMainnet.sol::L215, the comment `msg.sender must be owner.` should be `msg.sender must be SKALE Node address.`.
9. [FIXED] In MessageProxyForMainnet.sol::L365, `todo` and commented out code should be removed.
10. [FIXED] In MessageProxyForMainnet.sol::L286 `qual` -> `equal`.
11. [FIXED] In MessageProxyForMainnet.sol::L258 `Starning counter is not equal to incomin message counter` should be `Starting counter is not equal to incoming message counter`.

Adherence to Best Practices

1. [FIXED] In DepositBox.sol, the `GAS_AMOUNT_POST_MESSAGE * AVERAGE_TX_PRICE` value is used multiple times, but the constants themselves are never used seperately. It would be optimal to precalculate the value.
2. [FIXED] In EthERC20.sol, the function `_setupDecimals` is not used.
3. [FIXED] In MessageProxyForSchain.sol::L276-277 is redundant as L275 already ensures that it will never execute.
4. [FIXED] In LockAndDataForSchainERC20.sol, for consistency, `addERC20Token` should emit an event when a new ERC20 Token address is added.
5. [FIXED] In LockAndDataForSchainERC721.sol, for consistency, `addERC721Token` should emit an event when a new ERC721 Token address is added.
6. [FIXED] In LockAndDataForSchainERC20.sol, input validation in function `addERC20Token, addressERC20`.
7. [FIXED] In LockAndDataForSchainERC721.sol, input validation in function `addERC721Token, addressERC721`.
8. [FIXED] In TokenFactory.sol, input validation in function `constructor, erc20Module`.
9. In TokenManger.sol, input validation in function `constructor, newProxyAddress`.
10. [FIXED] In TokenManager.sol, to ensure consistent execution across all other functions, ensure that for all `contractThere` input for functions `[rawExitToMainERC20, rawTransferToSchainERC20, rawExitToMainERC721, rawTransferToSchainERC721]` and `to` for `[exitToMain, transferToSchain]`, to validate against zero address.
11. [FIXED] In LockAndDataForMainnetERC20.sol, input validation in function `sendERC20, contractHere` and `addERC20Token, addressERC20`.
12. [FIXED] In LockAndDataForMainnetERC721.sol, input validation in function `sendERC721, contractHere` and `addERC721Token, addressERC721`.
13. In MessageProxyForMainnet.sol, input validation in function `initialize, newContractManager` and `postOutgoingMessage, to`.

Test Results

Test Suite Results

We were able to run the tests successfully in both initial and reaudit stages.
The following results corresponds to the reaudit stage

```
  Contract: DepositBox
Your project has Truffle migrations, which have to be turn into a fixture to run your tests with Buidler
    tests for `deposit` function
       ✓ should rejected with `Unconnected chain` when invoke `deposit` (102ms)
       ✓ should rejected with `SKALE chain name is incorrect` when invoke `deposit` (66ms)
       ✓ should rejected with `Not enough money` when invoke `deposit` (148ms)
       ✓ should invoke `deposit` without mistakes (174ms)
       ✓ should revert `Not allowed. in DepositBox` (53ms)
    tests with `ERC20`
      tests for `depositERC20` function
         ✓ should rejected with `Not allowed ERC20 Token` (172ms)
         ✓ should invoke `depositERC20` without mistakes (394ms)
         ✓ should invoke `depositERC20` with some ETH without mistakes (379ms)
      tests for `rawDepositERC20` function
         ✓ should rejected with `Not allowed ERC20 Token` when invoke `rawDepositERC20` (167ms)
         ✓ should invoke `rawDepositERC20` without mistakes (348ms)
         ✓ should invoke `rawDepositERC20` with some ETH without mistakes (319ms)
    tests with `ERC721`
      tests for `depositERC721` function
         ✓ should rejected with `Not allowed ERC721 Token` (150ms)
         ✓ should invoke `depositERC721` without mistakes (291ms)
      tests for `rawDepositERC721` function
         ✓ should rejected with `Not allowed ERC721 Token` (146ms)
         ✓ should invoke `rawDepositERC721` without mistakes (281ms)
    tests for `postMessage` function
       ✓ should rejected with `Message sender is invalid` (57ms)
       ✓ should rejected with message `Receiver chain is incorrect` when schainID=`mainnet` (134ms)
       ✓ should rejected with message `Receiver chain is incorrect` when `sender != ILockAndDataDB(lockAndDataAddress).tokenManagerAddresses(schainHash)` (118ms)
       ✓ should rejected with message `Not enough money to finish this transaction` (134ms)
       ✓ should rejected with message `Invalid data` (186ms)
       ✓ should rejected with message `Could not send money to owner` (192ms)
       ✓ should transfer eth (212ms)
       ✓ should transfer ERC20 token (602ms)
       ✓ should transfer ERC20 for RAW mode token (854ms)
       ✓ should transfer ERC721 token (1042ms)
       ✓ should transfer RawERC721 token (606ms)

  Contract: ERC20ModuleForMainnet
     ✓ should invoke `receiveERC20` with `isRaw==true` (115ms)
     ✓ should invoke `receiveERC20` with `isRaw==false` (199ms)
     ✓ should return `true` when invoke `sendERC20` with `to0==address(0)` (597ms)
     ✓ should return `true` when invoke `sendERC20` with `to0==ethERC20.address` (285ms)
     ✓ should return `receiver` when invoke `getReceiver` with `to0==ethERC20.address` (246ms)
     ✓ should return `receiver` when invoke `getReceiver` with `to0==address(0)` (287ms)

  Contract: ERC20ModuleForSchain
     ✓ should invoke `receiveERC20` with `isRaw==true` (253ms)
     ✓ should rejected with `ERC20 contract does not exist on SKALE chain.` with `isRaw==false` (133ms)
     ✓ should invoke `receiveERC20` with `isRaw==false` (399ms)
     ✓ should return `true` when invoke `sendERC20` with `to0==address(0)` (572ms)
     ✓ should return send ERC20 token twice (521ms)
     ✓ should return `true` for `sendERC20` with `to0==address(0)` and `contractAddreess==address(0)` (442ms)
     ✓ should be rejected with incorrect Minter when invoke `sendERC20` with `to0==ethERC20.address` (593ms)
     ✓ should return true when invoke `sendERC20` with `to0==ethERC20.address` (668ms)
     ✓ should return `receiver` when invoke `getReceiver` with `to0==ethERC20.address` (394ms)
     ✓ should return `receiver` when invoke `getReceiver` with `to0==address(0)` (429ms)

  Contract: ERC721ModuleForMainnet
     ✓ should invoke `receiveERC721` with `isRaw==true`
     ✓ should invoke `receiveERC721` with `isRaw==false` (55ms)
     ✓ should return `true` when invoke `sendERC721` with `to0==address(0)` (459ms)
     ✓ should return `true` when invoke `sendERC721` with `to0==eRC721OnChain.address` (324ms)
     ✓ should return `receiver` when invoke `getReceiver` with `to0==eRC721OnChain.address` (124ms)
     ✓ should return `receiver` when invoke `getReceiver` with `to0==address(0)` (207ms)

  Contract: ERC721ModuleForSchain
```

```
    ✓ should invoke `receiveERC721` with `isRaw==true` (84ms)
    ✓ should rejected with `ERC721 contract does not exist on SKALE chain` with `isRaw==false` (127ms)
    ✓ should invoke `receiveERC721` with `isRaw==false` (390ms)
    ✓ should return `true` for `sendERC721` with `to0==address(0)` and `contractAddreess==address(0)` (502ms)
    ✓ should return `true` when invoke `sendERC721` with `to0==address(0)` (706ms)
    ✓ should return `true` when invoke `sendERC721` with `to0==eRC721OnChain.address` (377ms)
    ✓ should return `receiver` when invoke `getReceiver` with `to0==eRC721OnChain.address` (205ms)
    ✓ should return `receiver` when invoke `getReceiver` with `to0==address(0)` (467ms)

Contract: LockAndDataForMainnet
    ✓ should add wei to `lockAndDataForMainnet` (51ms)
    ✓ should check sendEth returned bool value (168ms)
    ✓ should work `sendEth` (112ms)
    ✓ should work `approveTransfer` (118ms)
    ✓ should work `getMyEth` (142ms)
    ✓ should rejected with `User has insufficient ETH` when invoke `getMyEth` (75ms)
    ✓ should rejected with `Not enough ETH. in `LockAndDataForMainnet.getMyEth`` when invoke `getMyEth` (131ms)
    ✓ should check contract without mistakes
    ✓ should rejected with `New address is equal zero` when invoke `getMyEth` (48ms)
    ✓ should rejected with `Contract is already added` when invoke `setContract` (47ms)
    ✓ should invoke addSchain without mistakes (91ms)
    ✓ should rejected with `SKALE chain is already set` when invoke `addSchain` (129ms)
    ✓ should rejected with `Incorrect Token Manager address` when invoke `addSchain` (64ms)
    ✓ should return true when invoke `hasSchain` (104ms)
    ✓ should return false when invoke `hasSchain`
    ✓ should invoke `removeSchain` without mistakes (216ms)
    ✓ should rejected with `SKALE chain is not set` when invoke `removeSchain` (160ms)

Contract: LockAndDataForMainnetERC20
    ✓ should rejected with `Not enough money` (71ms)
    ✓ should return `true` after invoke `sendERC20` (184ms)
    ✓ should return `token index` after invoke `addERC20Token` (369ms)

Contract: LockAndDataForMainnetERC721
    ✓ should NOT to send ERC721 to `to` when invoke `sendERC721` (145ms)
    ✓ should to send ERC721 to `to` when invoke `sendERC721` (240ms)
    ✓ should add ERC721 token when invoke `sendERC721` (135ms)

Contract: LockAndDataForSchain
    ✓ should set EthERC20 address (109ms)
    ✓ should set contract (356ms)
    ✓ should add schain (217ms)
    ✓ should add deposit box (210ms)
    ✓ should add gas costs (126ms)
    ✓ should remove gas costs (184ms)
    ✓ should reduce gas costs (397ms)
    ✓ should send Eth (282ms)
    ✓ should receive Eth (235ms)
    ✓ should return true when invoke `hasSchain` (67ms)
    ✓ should return false when invoke `hasSchain`
    ✓ should return true when invoke `hasDepositBox` (63ms)
    ✓ should return false when invoke `hasDepositBox`
    ✓ should invoke `removeSchain` without mistakes (174ms)
    ✓ should rejected with `SKALE chain is not set` when invoke `removeSchain` (98ms)
    ✓ should work `addAuthorizedCaller` (66ms)
    ✓ should work `removeAuthorizedCaller` (62ms)
    ✓ should invoke `removeDepositBox` without mistakes (107ms)
    ✓ should rejected with `Deposit Box is not set` when invoke `removeDepositBox` (50ms)

Contract: LockAndDataForSchain
    ✓ should set EthERC20 address (111ms)
    ✓ should set contract (439ms)
    ✓ should add schain (356ms)
    ✓ should add deposit box (252ms)
    ✓ should add gas costs (114ms)
    ✓ should reduce gas costs (409ms)
    ✓ should send Eth (246ms)
    ✓ should receive Eth (236ms)
    ✓ should return true when invoke `hasSchain` (72ms)
    ✓ should return false when invoke `hasSchain`
    ✓ should return true when invoke `hasDepositBox` (100ms)
    ✓ should return false when invoke `hasDepositBox`
    ✓ should invoke `removeSchain` without mistakes (115ms)
    ✓ should rejected with `SKALE chain is not set` when invoke `removeSchain` (130ms)
    ✓ should work `addAuthorizedCaller` (65ms)
    ✓ should work `removeAuthorizedCaller` (101ms)
    ✓ should invoke `removeDepositBox` without mistakes (120ms)
    ✓ should rejected with `Deposit Box is not set` when invoke `removeDepositBox` (59ms)

Contract: LockAndDataForSchainERC20
    ✓ should invoke `sendERC20` without mistakes (232ms)
    ✓ should rejected with `Amount not transfered` (69ms)
    ✓ should return `true` after invoke `receiveERC20` (218ms)
    ✓ should set `ERC20Tokens` and `ERC20Mapper` (106ms)

Contract: LockAndDataForSchainERC721
    ✓ should invoke `sendERC721` without mistakes (169ms)
    ✓ should rejected with `Token not transfered` after invoke `receiveERC721` (128ms)
    ✓ should return `true` after invoke `receiveERC721` (518ms)
    ✓ should set `ERC721Tokens` and `ERC721Mapper` (94ms)

Contract: MessageProxy
  MessageProxyForMainnet for mainnet
     ✓ should detect registration state by `isConnectedChain` function (210ms)
     ✓ should add connected chain (135ms)
     ✓ should remove connected chain (258ms)
     ✓ should post outgoing message (191ms)
     ✓ should post incoming messages (272ms)
     ✓ should get outgoing messages counter (181ms)
     ✓ should get incoming messages counter (499ms)
     ✓ should move incoming counter (160ms)
     ✓ should get incoming messages counter (734ms)
  MessageProxyForSchain for schain
     ✓ should detect registration state by `isConnectedChain` function (87ms)
     ✓ should add connected chain (124ms)
     ✓ should remove connected chain (237ms)
     ✓ should post outgoing message (269ms)
     ✓ should post incoming messages (474ms)
     ✓ should get outgoing messages counter (169ms)
     ✓ should get incoming messages counter (508ms)

Contract: TokenFactory
    ✓ should createERC20 (173ms)
    ✓ should createERC721 (203ms)

Contract: ERC20OnChain
    ✓ should invoke `totalSupplyOnMainnet`
    ✓ should rejected with `Call does not go from ERC20Module` when invoke `setTotalSupplyOnMainnet` (53ms)
    ✓ should invoke `setTotalSupplyOnMainnet` (103ms)
    ✓ should invoke `_mint` as internal (63ms)
    ✓ should invoke `burn` (107ms)
    ✓ should invoke `burnFrom` (157ms)

Contract: ERC721OnChain
    ✓ should invoke `mint` (71ms)
    ✓ should invoke `burn` (143ms)
    ✓ should reject with `ERC721Burnable: caller is not owner nor approved` when invoke `burn` (163ms)
    ✓ should invoke `setTokenURI` (106ms)

Contract: TokenManager
    ✓ should send Eth to somebody on Mainnet, closed to Mainnet, called by schain (305ms)
    ✓ should transfer to somebody on schain Eth and some data (566ms)
    ✓ should add Eth cost (428ms)
    ✓ should remove Eth cost (502ms)
    ✓ should rejected with `Not allowed ERC20 Token` when invoke `exitToMainERC20` (376ms)
    ✓ should rejected with `Not enough gas sent` when invoke `exitToMainERC20` (394ms)
    ✓ should invoke `exitToMainERC20` without mistakes (794ms)
    ✓ should rejected with `Not allowed ERC20 Token` when invoke `rawExitToMainERC20` (364ms)
    ✓ should rejected with `Not enough gas sent` when invoke `rawExitToMainERC20` (400ms)
    ✓ should revert `Not allowed. in TokenManager`
    ✓ should invoke `rawExitToMainERC20` without mistakes (733ms)
    ✓ should rejected with `Not allowed ERC20 Token` when invoke `transferToSchainERC20` (611ms)
    ✓ should invoke `transferToSchainERC20` without mistakes (706ms)
    ✓ should invoke `rawTransferToSchainERC20` without mistakes (715ms)
    ✓ should rejected with `Not allowed ERC20 Token` when invoke `rawTransferToSchainERC20` (639ms)
    ✓ should rejected with `Not allowed ERC721 Token` when invoke `exitToMainERC721` (674ms)
    ✓ should rejected with `Not enough gas sent` when invoke `exitToMainERC721` (668ms)
    ✓ should invoke `exitToMainERC721` without mistakes (831ms)
    ✓ should invoke `rawExitToMainERC721` without mistakes (719ms)
    ✓ should rejected with `Not allowed ERC721 Token` when invoke `rawExitToMainERC721` (686ms)
    ✓ should rejected with `Not enough gas sent` when invoke `rawExitToMainERC721` (671ms)
    ✓ should invoke `transferToSchainERC721` without mistakes (773ms)
    ✓ should rejected with `Not allowed ERC721 Token` when invoke `transferToSchainERC721` (652ms)
    ✓ should invoke `rawTransferToSchainERC721` without mistakes (759ms)
    ✓ should rejected with `Not allowed ERC721 Token` when invoke `rawTransferToSchainERC721` (646ms)
  tests for `postMessage` function
     ✓ should rejected with `Not a sender` (67ms)
     ✓ should be Error event with message `Receiver chain is incorrect` when schainID=`mainnet` (285ms)
     ✓ should be Error event with message `Invalid data` (273ms)
     ✓ should transfer eth (447ms)
     ✓ should rejected with `Incorrect receiver` when `eth` transfer (439ms)
     ✓ should transfer ERC20 token (855ms)
     ✓ should transfer rawERC20 token (1099ms)
     ✓ should transfer ERC721 token (849ms)
     ✓ should transfer rawERC721 token (845ms)
```

```
  186 passing (4m)
```

Code Coverage

Whilst there exists tests which provides code coverage up to a passable level, it is our strong recommendation that all code coverage be raised to the acceptable 100% level for all branches.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	94.35	71.08	92.86	92.65	
DepositBox.sol	100	78.79	100	100	
ERC20ModuleForMainnet.sol	97.3	70	100	97.37	105
ERC721ModuleForMainnet.sol	100	87.5	100	100	
LockAndDataForMainnet.sol	82.86	65.38	91.67	81.58	… 91,93,96,97
LockAndDataForMainnetERC20.sol	100	62.5	100	100	
LockAndDataForMainnetERC721.sol	100	62.5	100	100	
MessageProxyForMainnet.sol	89.55	63.89	94.12	84.72	… 535,536,539
PermissionsForMainnet.sol	62.5	50	50	60	56,57,71,82
contracts/interfaces/	100	100	100	100	
IContractManager.sol	100	100	100	100	
IERC20Module.sol	100	100	100	100	
IERC721Module.sol	100	100	100	100	
IMessageProxy.sol	100	100	100	100	
ISchainsInternal.sol	100	100	100	100	
contracts/predeployed/	86.74	70.36	85.16	85.79	
ERC20ModuleForSchain.sol	100	92.86	100	100	
ERC721ModuleForSchain.sol	100	91.67	100	100	
EthERC20.sol	92.16	57.14	90.48	92.16	191,192,210,215
LockAndDataForSchain.sol	82.72	80.77	96.43	83.53	… 349,350,352
LockAndDataForSchainERC20.sol	100	75	100	100	
LockAndDataForSchainERC721.sol	100	66.67	100	100	
MessageProxyForSchain.sol	63.29	54	63.64	61.63	… 456,457,460
OwnableForSchain.sol	66.67	62.5	83.33	72.73	66,77,86
PermissionsForSchain.sol	80	50	100	83.33	63
SkaleFeatures.sol	0	100	0	0	… 140,141,143
TokenFactory.sol	100	64.29	100	100	
TokenManager.sol	98.36	71.57	100	98.32	558,573
All files	**89.45**	**70.63**	**87.56**	**88.22**	

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

```
a581162e7409df3a9f5f72b4350eff7a1b4b0a46fdf155f6d5d832eaf82a514a ./IMA/proxy/contracts/PermissionsForMainnet.sol
c30eb440430f7a314575fa9e199186f4beca58ee13d8ab1b63893ea559928088 ./IMA/proxy/contracts/MessageProxyForMainnet.sol
56463900f833cca487f4624a7bdd66f6dae6ad046fd60f429e146cf1844b1ae1 ./IMA/proxy/contracts/ERC721ModuleForMainnet.sol
f958dc1a1a4915a88aa5af6a16c8195fe6de16487421ee625e073bd3aa5eaa0a ./IMA/proxy/contracts/ERC20ModuleForMainnet.sol
f68cb3d41c8632cca0a232513a188b947effef88dafe64651be8834b53d3c5c7 ./IMA/proxy/contracts/LockAndDataForMainnet.sol
c5af2b09f10e237cffd5f889849d63315531e900e6fd1d7a2f961c6c19ea6e06 ./IMA/proxy/contracts/DepositBox.sol
e3ff22a8995e628d1e05e99edcfc9a1a2016a337e06f9a4c69196233abce6e45 ./IMA/proxy/contracts/LockAndDataForMainnetERC20.sol
c4362ddc47d59c3cadb24fd1b02128ed57b86347c118355ac284a568a52326e0 ./IMA/proxy/contracts/LockAndDataForMainnetERC721.sol
595d60fef2ebc7636f86da980a0bc17b73a71aefc40d354583dbdc9e1dc85daf ./IMA/proxy/contracts/predeployed/LockAndDataForSchainERC721.sol
f767a8c51b9ce643bad75038b1fb967315cc95300abb4123327c3498af457889 ./IMA/proxy/contracts/predeployed/EthERC20.sol
dcd8e5cfcbedc8fbb57d89c5bafd73983c1b53af96bd8d9baac7999ad0ff0484 ./IMA/proxy/contracts/predeployed/ERC721ModuleForSchain.sol
a9d832d8379d078e3243c6d8d1dc5bf1b9da2a9f3fb1415742d6cb501a5b4553 ./IMA/proxy/contracts/predeployed/SkaleFeatures.sol
24ed25959a167758b740280ce2c764842e1e4f66d09e0b9aca3a38973e2d1f97 ./IMA/proxy/contracts/predeployed/TokenManager.sol
c40815ae415cb7195fa490bb1f210cf46d8a5f98090e0b80f86589a95787f0d7 ./IMA/proxy/contracts/predeployed/ERC20ModuleForSchain.sol
1c2ea3213b643a27989da484b1e710999a48ceed2e03a0a2f43ad851500ebe84 ./IMA/proxy/contracts/predeployed/OwnableForSchain.sol
0a7f8b0fc3633c649ee88720ddb5f3afda9e25c646ab2d815cc1ac52a82ded3f ./IMA/proxy/contracts/predeployed/MessageProxyForSchain.sol
0f6335e2b01d4d9eccada33da333b7bffd084f1277de28930bbf2d02443d4ae7 ./IMA/proxy/contracts/predeployed/PermissionsForSchain.sol
1dffd83fa2735b0b1300ddad511048b709d9961ae76fbba569b4dbd693bb1ce4 ./IMA/proxy/contracts/predeployed/LockAndDataForSchainERC20.sol
29880794a37dcac5ec49c10701f21bb6041dbdd06e38f0dd658bebfcebf473f2 ./IMA/proxy/contracts/predeployed/LockAndDataForSchain.sol
e9932454e8bd531e6d286a345272f6e91fa4a1a51bf957b6c22a5e5f36b0b065 ./IMA/proxy/contracts/predeployed/TokenFactory.sol
```

Tests

```
0c773f9f428d7653f3cb703db8b4837194c372323682b1853db3a7b0521867a0 ./IMA/proxy/contracts/test/TestSchains.sol
c1a6440a6517a7679d32397f564ad9d0da71a90f7ca6656cd3432fd55acf00a9 ./IMA/proxy/contracts/test/LockAndDataForMainnetWorkaround.sol
444018e4c5b9e392d9692a693aecc320d46acf2fedad1e0cf70acb586ba08a3e ./IMA/proxy/contracts/test/TestContractManager.sol
50164312e001184f94fd273b06a526a5b13d59bb1043b3b285c9576c22277199 ./IMA/proxy/contracts/test/LockAndDataForSchainWorkaround.sol
f736320870ae68daf01e28ef15fecc22012ad57bd211e8564cd66f55b91367d0 ./IMA/proxy/contracts/test/TestSchainsInternal.sol
550d7a3578e5b48ae010dd15f3d99a5829ab0ed78a09edb0649a668854ddef8a ./IMA/proxy/test/TokenFactory.spec.ts
22a0f39473f0037cf21988638eb5e93e57a10f3fd5fb107cd906054bf27f80ea ./IMA/proxy/test/LockAndDataForSchain.spec.ts
7dde350053fde8a66be59e4d2c843458057a257ac6db45281f0e125246f81e03 ./IMA/proxy/test/ERC20ModuleForSchain.spec.ts
a324105ee84e934b8b95231864d5247d97904f6a49fe14cba1554687ac2c96a6 ./IMA/proxy/test/ERC20ModuleForMainnet.spec.ts
c8520091ac471813239af2b2c29abfbd3ddbfb982a93a165ae36da411af82cde ./IMA/proxy/test/LockAndDataForSchainERC721.spec.ts
d5bea9f0badf80af6a6cd3db97c61563ff3db517dfaaf07aad52914b749c4b73 ./IMA/proxy/test/ERC721ModuleForMainnet.spec.ts
db738bce93d60527695f1b712f9a8adb4d9027a89e551aea3ea97e47ed2f4989 ./IMA/proxy/test/LockAndDataForSchain.ts
1cc7afa874135961b7d19f79fccf1bd298b95ac250b18dd2a4fa52a36db580f9 ./IMA/proxy/test/LockAndDataForMainnet.spec.ts
98829fbff58d80f7c01479ba683b1422297004059df06d4e6a0fe7f861cb29a5 ./IMA/proxy/test/MessageProxy.ts
7216cdccccd431b7cc69e22a033c665439cc5b4ecb9f19896f7b94f1c35adf4a ./IMA/proxy/test/LockAndDataForMainnetERC20.spec.ts
ede56d39f6e06dade0b2cba440958037f0786b3b292f18b4b5b3f493ab409bcb ./IMA/proxy/test/LockAndDataForSchainERC20.spec.ts
089a8b6b66c70ea027b275295152af0da87bfdf556754b2c7771eea9095f720e ./IMA/proxy/test/ERC721ModuleForSchain.spec.ts
38ab965ac85c122bcce1c81095394668d104d665e7205b5c1575824903635ff8 ./IMA/proxy/test/DepositBox.spec.ts
00bbcdc73dadad90a67d8e0d0ee2a0b88721bf52255105795b1d21dacd2306c1 ./IMA/proxy/test/LockAndDataForMainnetERC721.spec.ts
47d4300744251d5e525a57c1f3ef9bebbd7d47179aca60befa4f13fad5c27634 ./IMA/proxy/test/TokenManager.spec.ts
e44d967443fcc1efaf477308fca44c5cf85618f0d08c7bda09fdd448e40b8d53 ./IMA/proxy/test/utils/helper.ts
dd3c4dd574f0aea1c9854c428d768f9d3e1ae579a5e4f1e44fd5cb128039784e ./IMA/proxy/test/utils/command_line.ts
f6876bdbdcb522a00e3672b9ebfc3a5f9f6f4823c0dec67c7d62fa3b9c59f7de ./IMA/proxy/test/utils/time.ts
319269694633baacde87d3d7178888d172812f01e714646435f22d0673b2a599 ./IMA/proxy/test/utils/deploy/lockAndDataForMainnetERC721.ts
259298babbc37f7f980911a37716cb0d1deb94f8a2f5827c65a35d2b6314e866 ./IMA/proxy/test/utils/deploy/lockAndDataForMainnet.ts
9c6e1427ab9a7dd7c9a111746fd1c4e7740e56a4cfaec849951fdc42f33cb934 ./IMA/proxy/test/utils/deploy/messageProxyForMainnet.ts
13eac3123265b210d829cba03c1c0c5901aa4f9a2982a16796c049001d5f1a51 ./IMA/proxy/test/utils/deploy/erc721ModuleForMainnet.ts
231ebc8de6d392832df586df7dfd8623f344fb6058f6f056f76e41311aa54e31 ./IMA/proxy/test/utils/deploy/erc20ModuleForMainnet.ts
87d2f2f0ced1a0ea5cece4c6f1ffee429f4510e5c34cd8eb00b8ce2c45fa9ab4 ./IMA/proxy/test/utils/deploy/depositBox.ts
b99061587a7ca6579456fdaca85439d40b20005d9174032773d7f1a62f19c0d8 ./IMA/proxy/test/utils/deploy/lockAndDataForMainnetERC20.ts
```

Changelog

- 2020-11-25 - Initial report
- 2021-01-08 - Reaudit commit taken at ee72736
- 2021-01-14 - Tests results updated along with coverage. Also explicitly stated Best Practices and Documentation statuses when fixed or mitigated, along with adding one new documentation issue and removing stamp for addressing all best practices.

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.