Bridge

SMART CONTRACT AUDIT

ZOKYO.

May 18th, 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

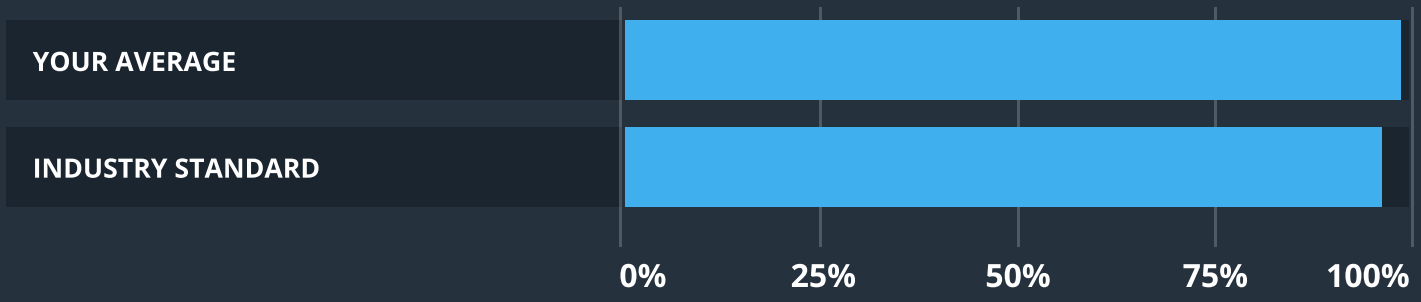
TECHNICAL SUMMARY

This document outlines the overall security of the BridgeNetwork smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the BridgeNetwork smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code is 98%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the BridgeNetwork team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the BridgeNetwork repository.

Repository - https://github.com/bridgeNetwork1/bridgeContractV2

Last commit - a167d62c5862e950ab21cd64aea450d1ad901747

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- Bridge.sol
- BridgePool.sol
- Controller.sol
- Deployer.sol
- FeeController.sol
- Registry.sol
- Settings.sol
- Token.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of BridgeNetwork smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were no critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

Contracts are well written and structured. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

Despite the fact, the expected logic is managing all vestings by the owner, it should be careful with parameters to avoid mistakes during the vesting process.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

CRITICAL | RESOLVED

In contract "Controller", in function "addRegistrar", at lines 65 and 67, "validators.length" was used instead of "registrars.length".This leads to improper deletion of some values from the "registrars".

Recommendation:
Through iteration from line 65, replace “validators.length” with “registrars.length”.

MEDIUM | RESOLVED

In contract bridge.sol, functions send and burn do not implement the re-entrancy protection properly, you will think it is not necessary to specify the nonRenetrat modifier to the send and burn functions because those both functions calls the deductFee function which have that protection active, but however before calling the deductFee function, there is another external call to the processedPayment function at line 423, here an malicious user, could create a token that will have a modified allowance function and with that will be possible to re-entar the send or burn functions, we are aware that the tokens addresses need to be pre-approved by an administrator but the malicious code could hide in plain sight and the malicious actor could use this technique to extract liquidity from his community using bridge.sol contract.

Recommendation:
Add the nonReentrant modifier to all the public/external functions that are doing external calls or are changing the contract state and remove it from the internal/private functions, the internal/private functions will be protected from re-entracy because the only way to call them is through an public/external function anyway, so there is no need to add nonReentrant modified on the private/internal function, but it is a strong need to add them on all the external/public ones that are doing external calls or are changing the contract state.

MEDIUM | RESOLVED

In contract bridge.sol, function processedPayment, line 424, is a call to the transferFrom function from a ERC20 contract, because the address of the token contract is a token with user input, this can lead to a malicious attack, not all the tokens have requires inside their transfer or transferFrom functions that will make them fail if something is not ok during execution because the ERC20 standard does not requires it and this is the reason why the transfer functions have a return true statement.

Recommendation:
Add SafeERC20 library in the contract and use safeTransferFrom instead of transferFrom to make interactions with any tokens safe.

MEDIUM | RESOLVED

In contract tokenLock.sol, function processedPayment, there's a transferFrom call from a ERC20 contract without checking whether the call was succesful.

Recommendation:
Add SafeERC20 library in the contract and use safeTransferFrom instead of transferFrom to make interactions with any tokens safe.

INFORMATIONAL | UNRESOLVED

When running the coverage tool with the solidity optimizer disabled it results in a stack too deep error when compiling. This happens because the EVM is limited to assigning 16 slots for local variables. Running with optimizer solves the issue but most of the external plugins, like solidity-coverage, does not support the optimizer, to have better access to tolling and plugins we will recommend to fix the issue.

```
Compilation:
============

CompilerError: Stack too deep when compiling inline assembly: Variable headStart is 1 slot(s) too deep inside the stack.

Error in plugin solidity-coverage: HardhatError: HH600: Compilation failed
```

Recommendation:
Refactor functions that have many local variables and try to split them in smaller and more manageable functions.

INFORMATIONAL | RESOLVED

In contract settings.sol there's an assembly block at lines 62-64. This also applies to the bridge.sol file where there's the same assembly block inside the constructor.

Recommendation:
Extract the assembly block in a separate method and add a disable comment for the no-inline-assembly warning. This way you don't mix assembly methods with the code logic.

INFORMATIONAL | RESOLVED

When iterating consider declaring the length variable as a local variable instead of reading it each time inside the loop. This helps save gas costs, as the most expensive operations are storage reads/writes, so by reading the variable only once it reduces the gas cost. For example in contract settings.sol, at line 67 the for loop can be refactored as follows. Do this for all for loop occurrences.

INFORMATIONAL | RESOLVED

There are multiple versions of solidity in different files. For example in tokenLock.sol the version declared is pragma solidity ^0.8.2,but the declared compiler version in hardhat.config is 0.8.0.

Recommendation:
Use consistent versioning.

INFORMATIONAL | RESOLVED

There are multiple contracts where global variables have no explicit visibility declarations. For example in the settings.sol file at line 17 there's no visibility declared.

Recommendation:
Add explicit visibility declarations in all contracts and consider refactoring global variables in groups, such as public and private groups.

INFORMATIONAL | RESOLVED

There is mixed usage of error messages when reverted inside functions. Some error messages are incomplete or plain abbreviations that are explained inside errorsMsgs.txt. That's not the preferred way to have error messages.

Recommendation:
Remove the .txt file and add full length error messages whenever there's require.

INFORMATIONAL | RESOLVED

Most of the .sol files in the contract directory are lower case, so are the names of the contracts. It's preferred that contract names start with capital letters and follow a camelCase style.

Recommendation:
Refactor the contract names and if you wish also rename the .sol files to match the contract casing. Also most of the code is not formatted correctly or using inconsistent indentation. Use a tool such as solhint and format/indent the code correctly as it's hard to follow. Also, for further details check: https://docs.soliditylang.org/en/v0.8.11/style-guide.html

INFORMATIONAL | RESOLVED

In contract "Wrapped Deployer", the variable "lossessEnabled" is initialized when it is declared at line 11 and there is no way to modify it.

Recommendation:
The variable should be initialized through the constructor and a function should be added to allow the variable to be modified.

	Bridge.sol	BridgePool.sol	Controller.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	Deployer.sol	FeeController.sol	Registry.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	Settings.sol	Token.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by BridgeNetwork team

As part of our work assisting BridgeNetwork in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the BridgeNetwork contract requirements for details about issuance amounts and how the system handles these.

```
Contract: Controller
  addAdmin
    ✓ call not from owner
    ✓ addAdmin with _admin = a valid address and add = true (39ms)
    ✓ addAdmin with _admin = used address and add = true
    ✓ addAdmin with _admin = not an admin and add = false
    ✓ addAdmin with _admin = a valid address and add = false (62ms)
  addRegistrar
    ✓ call not from admin
    ✓ addRegistrar with _registrar = a valid address and add = true
    ✓ addRegistrar with _registrar = used address and add = true
    ✓ addRegistrar with _registrar = not an registrar and add = false
    ✓ addRegistrar with _registrar = a valid address and add = false (70ms)
  addOracle
    ✓ call not from admin
    ✓ addOracle with _oracle = a valid address and add = true
    ✓ addOracle with _oracle = used address and add = true
    ✓ addOracle with _oracle = not an oracle and add = false
    ✓ addOracle with _oracle = a valid address and add = false (60ms)
  addValidator
    ✓ call not from admin
    ✓ addValidator with _validator = a valid address and add = true
    ✓ addValidator with _validator = used address and add = true
    ✓ addValidator with _validator = not an oracle and add = false
    ✓ addValidator with _validator = a valid address and add = false (62ms)
Contract: Deployer
  update bridge
    ✓ caller is not the owner (43ms)
```

✓ invalid address
✓ valid address
updateLossless
✓ caller is not the admin (263ms)
✓ valid address
deployerWrappedAsset
✓ bridge is zero address
✓ losseless = 1 (62ms)
✓ losseless != 1 (48ms)

Contract: RegistryContract

completeSendTransaction
✓ invalid Transaction (38ms)
✓ set isCompleted true (68ms)
completeBurnTransaction
✓ invalid Transaction
✓ set isCompleted true (48ms)
completeMintTransaction
✓ invalid Transaction
✓ set isCompleted true (248ms)
completeMintTransaction
✓ invalid Transaction
✓ set isCompleted true (64ms)
transactionValidated
✓ tx is not validated
registerTransaction
✓ send Transaction
✓ burn Transaction
✓ other Transaction
registerClaimTransaction
✓ invalid controller (44ms)
✓ register already registered claim transaction (61ms)
✓ isAssetSupportedChain false (41ms)
✓ invalid claim ID (44ms)
✓ success register claim transaction (49ms)
registerMintTransaction
✓ invalid controller (50ms)
✓ register already registered mint transaction (211ms)
✓ invalid wrappedAddress (175ms)
✓ foriegn asset not set (495ms)
✓ chain error (200ms)
✓ invalid mind ID (269ms)

✓ success register mint transaction (205ms)
validateTx
✓ invalid controller and settings (40ms)
✓ mintable = true
✓ mintable = false
✓ incomplet transaction
✓ update valid sigs (191ms)
getEthSignedMessageHash
✓ expectMessage is not empty string
getSigner
✓ valid signature
splitSignature
✓ invalid signature
Contract: FeeController
activateBrgHoldingIncentive
✓ caller is not the admin (49ms)
✓ with same status
✓ change status to true
activateAssetIncentive
✓ caller is not the admin
✓ with same status
✓ change status to true
activateAddressExemption
✓ caller is not the admin
✓ with same status
✓ change status to true
exemptAddress
✓ caller is not the owner
✓ with same status
✓ change status to true
setAssetIncentivization
✓ caller is not the owner
✓ with same value
✓ incentive + brgHoldingIncentive is above limit
✓ modify incentive
setBrgHoldingThreshold
✓ caller is not the admin
✓ with same status
✓ change threshold (276ms)
setAssetIncentivization
✓ caller is not the owner
✓ with same value

```
    ✓ incentive + brgHoldingIncentive is above limit
    ✓ modify incentive
  setBrgHoldingThreshold
    ✓ caller is not the admin
    ✓ with same status
    ✓ change threshold
  setBrgHoldingIncentive
    ✓ caller is not the owner
    ✓ with same value
    ✓ incentive is above limit
    ✓ modify incentive
  getBridgeFee
    ✓ user is exempted (50ms)
    ✓ usebrgHoldingIncentive= true and sender balance >= brgHoldingThreshold (48ms)
    ✓ usebrgHoldingIncentive= true and sender balance < brgHoldingThreshold (68ms)
    ✓ useAssetIncentive is true and assetIncentive[asset] > 0 (126ms)
    ✓ totalIncentive grather then 100 (56ms)
  getIncentive
    ✓ incentive is 0
    ✓ incentive and fee != 0
Contract: Bridge
    ✓ should check bridge deployment
    ✓ should pause bridge
    ✓ should active native asset, all checks
    ✓ should update asset, all checks
    ✓ should register rail, all checks
    ✓ should add foreign asset, all checks
    ✓ should send asset, all checks
    ✓ should burn asset, all checks
    ✓ should mint asset, all checks
    ✓ should claim asset, all checks
    ✓ should do initial migrations, all checks
    ✓ should do complete migration, all checks
    ✓ should migrate foreign asset, all checks
    ✓ should migrate native asset, all checks
    ✓ should register new native migration, all checks
    ✓ should register foreign migration, all checks
Contract: BridgeToken
  checks
    ✓ has a name (45ms)
    ✓ has a symbol
    ✓ has 18 decimals
```

✓ has minted supply
✓ has correct initial balance for deployer
✓ has 0 initial allowance for deployer

contract

✓ should mint to address
✓ should revert mint for zero address (38ms)
✓ should burn amount
✓ should not burn amount for zero address
✓ should not burn if amount greater than balance
✓ should blacklist address
✓ should be able to blacklist only once
✓ should remove blacklist if blacklisted (40ms)
✓ should transfer funds
✓ should revert transfer for wrong inputs (108ms)
✓ should transferFrom if has allowance
✓ should revert transferFrom if address blacklisted
✓ should revert transferFrom if not enough allowance
✓ should not spend allowance if allowance is max uint256 (43ms)
✓ should approve and increase allowance
✓ should revert approve for owner address zero
✓ should revert approve for zero spender
✓ should increase allowance
✓ should decrease allowance (83ms)

Contract: Settings

checks

✓ should update fee
✓ should revert update for 0 fee
✓ shoud not update for same fee twice (39ms)
✓ should not update is chain not supported
✓ should revert update if caller not admin
✓ should revert update if caller not owner
✓ should set railOwnerFeeShare
✓ should revert set railOwnerFeeShare for sameVal (43ms)
✓ should revert set railOwnerFeeShare for incorrect share
✓ should set updatableAssetState
✓ should revert set updatableAssetState for same val
✓ should set onlyOwnableRail
✓ should revert set onlyOwnableRail for same val
✓ should set railRegistrationFee
✓ should revert set railRegistrationFee for same val (40ms)
✓ should set feeRemitance address

```
    ✓ should revert set feeRemitance address for caller not owner
    ✓ should revert set feeRemitance for address zero
    ✓ should revert set feeRemitance for address same val
    ✓ should return correct minValidations
    ✓ returns 0 for 0 validators count
    ✓ should set approvedToAdd
    ✓ should not set approvedToAdd for same status (48ms)
    ✓ should return empty supported chains after deploy
    ✓ should set validation percentage (40ms)
    ✓ should not set validation percentage for caller not authorized
    ✓ should not set validation percentage for same percentage
    ✓ should not set validation percentage for wrong percentage
    ✓ should set bridge token
    ✓ should not set bridge token for zero address
    ✓ should set min withdrawable fee
    ✓ should not set 0 min withdrawable fee
    ✓ should add supported chains (271ms)
    ✓ should revert for chains and fee len not match
    ✓ should not add current chain id to supported (48ms)
    ✓ should remove supported chain ids (73ms)
    ✓ should work when called to remove but no supported chains (244ms)
Contract: Token
  checks
    ✓ owner should have correct balance after deploy
    ✓ has name and symbol
Contract: Settings
  checks
    ✓ should harvest - read once
    ✓ should harvest - read every time
Contract: TokenLock
  checks
    ✓ should add lock (69ms)
    ✓ should revert add lock for invalid inputs
    ✓ should revert lockAsset for invalid lock id
    ✓ should revert lockAsset for amount lower than min
    ✓ should lockAsset (78ms)
    ✓ should revert lockAsset for no allowance (41ms)
    ✓ should harvest
    ✓ should revert harvest if completed
    ✓ should revert harvest if not yet time
    ✓ should harvest and update times
    ✓ should revert harvest for excess amountss
```

```
190  passing (60s)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
BridgePool.sol	100	100	100	100	
Controller.sol	100	100	100	100	
Deployer.sol	100	100	100	100	
FeeController.sol	100	100	100	100	
Registry.sol	100	100	100	100	
Settings.sol	100	100	100	100	
Token.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

We are grateful to have been given the opportunity to work with the BridgeNetwork team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the BridgeNetwork team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.