

HAECHI AUDIT

CrossChainBridge

Smart Contract Security Analysis

Published on : Jan 28, 2022

Version v1.0





HAECHI AUDIT

Smart Contract Audit Certificate



CrossChainBridge

Security Report Published by HAECHI AUDIT
v1.0 Jan 28, 2022

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	1	-	-	-	-
Major	-	-	-	-	-
Minor	2	-	-	-	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

3 Issues (1 Critical, 0 Major, 2 Minor) Found

[TABLE OF CONTENTS](#)

[ABOUT US](#)

[INTRODUCTION](#)

[SUMMARY](#)

[OVERVIEW](#)

[FINDINGS](#)

[Front running attack is possible using the CrossChainBridgeERC20V1#releaseNative\(\) function. _____
\(Found - v1.0\).](#)

[It is advised to define the wrappedNative\(\) function of the router as a view function.](#)

[The parameter of the LiquidityRemoved event that occurs in the
CrossChainBridgeERC20LiquidityManagerV1#_withdrawLiquidityERC20\(\) function may fail to return a
normal value.](#)

[DISCLAIMER](#)

[Appendix A. Test Results](#)

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring.

We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe, 1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries : audit@haechi.io

Website : audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of the smart contract created by CrossChainBridge team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by CrossChainBridge team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

 CRITICAL

Critical issues must be resolved as critical flaws that can harm a wide range of users.

 MAJOR

Major issues require correction because they either have security problems or are implemented not as intended.

 MINOR

Minor issues can potentially cause problems and therefore require correction.

 TIPS

Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends that CrossChainBridge team improve all issues discovered. The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, and *Sample#fallback()* means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub

(<https://github.com/HAECHI-LABS/CrosschainBridge-audit/tree/de43c5b8e2c8fbf4690f04a50b44fe02d73b8d6d>). The last commit of the code used for this Audit is “de43c5b8e2c8fbf4690f04a50b44fe02d73b8d6d”.

Issues HAECHI AUDIT found 1 critical issue, 0 major issues, and 2 minor issues. There are 0 Tips issue explained that would improve the code’s usability or efficiency upon modification

Severity	Issue	Status
 CRITICAL	Front running attack is possible using the CrossChainBridgeERC20V1#releaseNative() function.	(Found - v1.0)
 MINOR	It is advised to define the wrappedNative() function of the router as a view function.	(Found - v1.0)
 MINOR	The parameter of the LiquidityRemoved event that occurs in the CrossChainBridgeERC20LiquidityManagerV1 #_withdrawLiquidityERC20() function may fail to return a normal value.	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- ❖ BridgeChefV1
- ❖ BuyBackAndBurn
- ❖ CrossChainBridgeERC20LiquidityManagerV1
- ❖ CrossChainBridgeERC20V1
- ❖ CrossChainBridgeERC721V1
- ❖ LiquidityMiningPoolsV1
- ❖ MultiSignatureOracleV1
- ❖ RewardPoolsV1
- ❖ RouterBSCPancakeV1
- ❖ RouterETHUniswapV1
- ❖ RouterPOLYSushiV1
- ❖ Bridge
- ❖ MintableERC20
- ❖ MintableERC721
- ❖ PoolsInterestBearingToken
- ❖ MyPausable
- ❖ MyPausableUpgradeable

FINDINGS

CRITICAL

Front running attack is possible using the `CrossChainBridgeERC20V1#releaseNative()` function. (Found - v.1.0)

```
/**
 * @notice Releases native tokens in this network that were deposited in another
network
 *
 * (effectively completing a bridge transaction)
 *
 * @param sigV Array of recovery Ids for the signature
 * @param sigR Array of R values of the signatures
 * @param sigS Array of S values of the signatures
 * @param receiverAddress The account to receive the tokens
 * @param sourceNetworkTokenAddress the address of the ERC20 contract in the network
the deposit was made
 * @param amount The amount of tokens to be released
 * @param depositChainId chain ID of the network in which the deposit was made
 * @param depositNumber The identifier of the corresponding deposit
 * @dev emits event TokensReleased after successful release
 */
function releaseNative(
    uint8[] memory sigV,
    bytes32[] memory sigR,
    bytes32[] memory sigS,
    address receiverAddress,
    address sourceNetworkTokenAddress,
    uint256 amount,
    uint256 depositChainId,
    uint256 depositNumber
) external nonReentrant {
    // release wrapped native (ERC20) tokens from bridge without transferring them
    // tokens will be swapped from ERC20 to native token in the following steps
    uint256 releaseAmountAfterFees = _releaseERC20(
        sigV,
        sigR,
        sigS,
        receiverAddress,
        sourceNetworkTokenAddress,
        amount,
```

```

        depositChainId,
        depositNumber,
        true
    );

    // check native token balance before ERC20-to-native swap
    uint256 contractBalance = address(this).balance;

    // swap wrapped native ERC20 token back to native token
    wrappedNative.withdraw(releaseAmountAfterFees);

    // check if native token balance has increased by release amount
    require(
        address(this).balance == contractBalance + releaseAmountAfterFees,
        'CrossChainBridgeERC20: error while unwrapping native tokens'
    );

    // send native token back to user
    payable(_msgSender()).transfer(releaseAmountAfterFees);
}

```

[<https://github.com/HAECHI-LABS/CrosschainBridge-audit/blob/de43c5b8e2c8fbf4690f04a50b44fe02d73b8d6d/contracts/CrossChainBridgeERC20V1.sol#L241-L293>]

Issue

The `CrossChainBridgeERC20V1#releaseNative()` deposits the native token of network A in the network and then withdraws the native token in network B via the signature of multi oracle. However, the multi oracle signature contains the receiver address, it sends the native token of the current network to msg.sender when the signature is verified.

Thus, when a user who actually intends to move a native token between networks through the bridge sends the `CrossChainBridgeERC20V1#releaseNative()` function to txPool, another user can perform a front running attack by initiating the transaction first to obtain the native token.

Recommendation

We recommend changing the logic to send the native token of the current network to the receiver address included in the signature if the signature verification from multi oracle is completed.

◉ MINOR

It is advised to define the `wrappedNative()` function of the router as a view function.
(Found - v.1.0)

```
/**
 * @notice Returns the address of the wrapped native token that is used by the dex
 */
function wrappedNative() external override returns (address) {
    return pancakeRouter.WETH();
}
```

[<https://github.com/HAECHI-LABS/CrosschainBridge-audit/blob/de43c5b8e2c8fbf4690f04a50b44fe02d73b8d6d/contracts/router/RouterBSCPancakeV1.sol#L78-L83>]

```
/**
 * @notice Returns the address of the wrapped native token that is used by the dex
 router
 */
function wrappedNative() external override returns (address) {
    return uniswapRouter.WETH();
}
```

[<https://github.com/HAECHI-LABS/CrosschainBridge-audit/blob/de43c5b8e2c8fbf4690f04a50b44fe02d73b8d6d/contracts/router/RouterETHUniswapV1.sol#L78-L83>]

```
/**
 * @notice Returns the address of the wrapped native token that is used by the dex
 */
function wrappedNative() external override returns (address) {
    return sushiSwapRouter.WETH();
}
```

[<https://github.com/HAECHI-LABS/CrosschainBridge-audit/blob/de43c5b8e2c8fbf4690f04a50b44fe02d73b8d6d/contracts/router/RouterPOLYSushiV1.sol#L78-L83>]

Issue

The `RouterBSCPancakeV1#wrappedNative()`, `RouterETHUniswapV1#wrappedNative()`, and `RouterPOLYSushiV1#wrappedNative()` functions are external functions that return

the address of the wrapped native token in each network. However, because they are not declared as a view function, the return value cannot be obtained.

Recommendation

We advise adding a statement to request a claim to the connected pool even when the `DCBVault#withdraw()` function is called.

• MINOR

The parameter of the LiquidityRemoved event that occurs in the `CrossChainBridgeERC20LiquidityManagerV1#_withdrawLiquidityERC20()` function may fail to return a normal value.

(Found - v.1.0)

```

/**
 * @notice Removes ERC20 liquidity from a pool
 *
 * Private interface to this function which allows internal calls from
 * reentrancy guard protected functions
 *
 * @param token the token for which liquidity should be removed from this pool
 * @param amount the amount of liquidity to be removed
 * @dev emits event LiquidityRemoved
 */
function _withdrawLiquidityERC20(IERC20 token, uint256 amount)
    private
    whenNotPaused
    returns (uint256 withdrawalAmount)
{
    require(amount > 0, 'LiquidityManager: amount cannot be 0');

    // check if liquidity is sufficient for withdrawal
    require(token.balanceOf(address(bridgeERC20)) ≥ amount, 'LiquidityManager: not
    enough liquidity in bridge');

    // determine the fee rate to be used for this transaction (usually default liquidity
    withdrawal fee)
    uint256 liquidityWithdrawalFee = defaultLiquidityWithdrawalFee;
    // if a specific fee rate is stored for this particular release token then we use
    this rate instead
    if (liquidityWithdrawalFees[address(token)] > 0) {
        liquidityWithdrawalFee = liquidityWithdrawalFees[address(token)];
    }

    // calculate the fee amounts (dividing by 1.000.000 since the fee rate is provided as
    parts per million [ppm])
    // calculate the total fee amount for this transaction
    uint256 withdrawalFeeAmount = (amount * liquidityWithdrawalFee) / 1000000;
    // calculate the remaining amount that will be released to the user
    withdrawalAmount = amount - withdrawalFeeAmount;

    // transfer developer account fee, if applicable
    if (devAddr != address(0) && withdrawalFeeAmount > 0) {
        token.safeTransferFrom(address(bridgeERC20), devAddr, withdrawalFeeAmount);
    }
}

```

[<https://github.com/HAECHI-LABS/CrosschainBridge-audit/blob/de43c5b8e2c8fbf4690f04a50b44fe02d73b8d6d/contrac>

Issue

The `CrossChainBridgeERC20LiquidityManagerV1#_withdrawLiquidityERC20()` function is called from the functions

`CrossChainBridgeERC20LiquidityManagerV1#withdrawLiquidityNative()` and

`CrossChainBridgeERC20LiquidityManagerV1#withdrawLiquidityERC20()`, and is also a

function that withdraws some of the deposited tokens. In the case of a deposit, liquidity is added as much as the amount of the deposited token, and the amount parameter of the LiquidityRemoved event is also entered as the amount of the deposited token.

However, for withdrawal, the amount excluding the fee from the withdrawal amount enters the parameter of the LiquidityRemoved event even though liquidity is removed by the amount of tokens to be withdrawn

Recommendation

We recommend modifying the event to occur with the amount including withdrawalFee in the amount parameter of the LiquidityRemoved event.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on Klaytn. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to existing issues.

BridgeChefV1

#initialize()

- ✓ should fail if dev address is ZERO ADDRESS
- ✓ should fail if staking address is ZERO ADDRESS

after initialization

#add()

- ✓ should fail if contract is paused
- ✓ should fail if msg.sender does not have FARM_ADMIN_ROLE
- ✓ should fail if lpToken address is ZERO ADDRESS
- ✓ should fail lpToken is already used in farm

valid case

- ✓ lpToken marked as used
- ✓ farm information update
- ✓ should emit FarmAdded event

#updateFarmMultiplier()

- ✓ should fail if msg.sender does not have FARM_ADMIN_ROLE
- ✓ should fail if try to update non-existent farm

valid case

- ✓ farm information update
- ✓ should emit FarmMultiplierChanged event

#deposit()

- ✓ should fail if contract paused
- ✓ should fail if farmId >= farmInfo.length
- ✓ should fail if deposit amount is 0
- ✓ should fail if msg.sender does not approve contract

valid case

- ✓ lpToken move to chef contract
- ✓ user deposit information update
- ✓ should emit DepositAdded event

#harvest()

- ✓ should fail if contract is paused
- ✓ should fail if invalid farm Id

- ✓ should fail if invalid amount
- valid case
 - ✓ user earn bridge token
 - ✓ should emit RewardsHarvested event

#withdraw()

- valid case
 - ✓ user get deposited lpToken
 - ✓ user earn bridge token
 - ✓ should emit RewardsHarvested event
 - ✓ should emit FundsWithdrawn event

setter function

- ✓ should fail if msg.sender does not have appropriate role (62ms)
- ✓ should fail if sanity check fail (41ms)

valid case

- ✓ parameter change properly

CrossChainBridgeERC20LiquidityManagerV1

#depositERC20()

- ✓ should fail if amount is zero
- ✓ should fail if invalid token address
- ✓ should fail if try to deposit more than user balance
- ✓ should fail if user does not approve buyback contract
- ✓ should fail if paused

valid case

- ✓ update collected information
- ✓ should emit TokensAdded event

#depositNativeToken()

- ✓ should fail if amount is zero
- ✓ should fail if msg.value does not match the amount
- ✓ should fail if paused

valid case

- ✓ receive function acts like depositNativeToken
- ✓ update collected information
- ✓ should emit TokensAdded event

#buybackAndBurnNative()

- ✓ should fail if collected is zero (11278ms)
- ✓ should fail if paused

valid case

- ✓ token burned
- ✓ collected information clear
- ✓ should emit BoughtBackAndBurned

#buybackAndBurnERC20()

- ✓ should fail if collected is zero (3079ms)

- ✓ should fail if paused

valid case

- ✓ token burned

- ✓ collected information clear

- ✓ should emit BoughtBackAndBurned

setter function

- ✓ should fail if msg.sender does not have appropriate role (94ms)

- ✓ should fail if sanity check fail (64ms)

valid case

- ✓ parameter change properly

CrossChainBridgeERC20LiquidityManagerV1

#addLiquidityNative()

- ✓ should fail if msg.value does not match the amount

- ✓ should fail if paused (1421ms)

valid case

- ✓ deposit user get lpToken of wrappedNative

- ✓ should emit LiquidityAdded event

- ✓ should emit LiquidityPoolCreated event if bridgeERC20's lpToken does not exist

#addLiquidityERC20()

- ✓ should fail if deposit user does not approve manager contract

- ✓ should fail if deposit user does not have enough token to deposit

valid case

- ✓ deposit user get lpToken of wrappedNative

- ✓ should emit LiquidityAdded event

- ✓ should emit LiquidityPoolCreated event if bridgeERC20's lpToken does not exist

#withdrawLiquidityNative()

- ✓ should fail if not enough liquidity

- ✓ should fail if deposit user does not approve lpToken to manager contract

- ✓ should fail if user try to withdraw more than deposit amount

valid case

- ✓ user get native token (636ms)

- ✓ if dev address set, withdrawal fee transfer to dev address

- 1) should emit LiquidityRemoved event

#withdrawLiquidityERC20()

- ✓ should fail if not enough liquidity

- ✓ should fail if deposit user does not approve lpToken to manager contract

- ✓ should fail if user try to withdraw more than deposit amount

valid case

- ✓ user get deposited token
- ✓ user lpToken burned
- ✓ if dev address set, withdrawal fee transfer to dev address

2) should emit LiquidityRemoved event

setter function

- ✓ should fail if msg.sender does not have appropriate role (94ms)
- ✓ should fail if sanity check fail (64ms)

valid case

- ✓ parameter change properly

CrossChainBridgeERC20V1

#depositNative()

- ✓ should fail if msg.value does not match the amount
- ✓ should fail if paused (875ms)

valid case

- ✓ should emit TokensDeposited event

#depositERC20()

- ✓ should fail if deposit user does not approve manager contract
- ✓ should fail if deposit user does not have enough token to deposit

valid case

- ✓ should emit TokensDeposited event

#releaseNative()

- ✓ should fail if released was already processed
- ✓ should fail if invalid parameter given
- ✓ should fail if invalid signature

valid case

3) user get another network native coin

- ✓ fee transfer to buyBackAndBurn/liuquidityMiningPools/rewardPools (if fee set) (5343ms)

#releaseERC20()

- ✓ should fail if released was already processed
- ✓ should fail if invalid parameter given
- ✓ should fail if invalid signature

valid case

- ✓ user get another network token

- ✓ fee transfer to buyBackAndBurn/liuquidityMiningPools/rewardPools (if fee set)

setter function

- ✓ should fail if msg.sender does not have appropriate role (94ms)
- ✓ should fail if sanity check fail (64ms)

valid case

- ✓ parameter change properly

CrossChainBridgeERC20LiquidityManagerV1

#deposit()

✓ should fail collection address does not registered in official (only when target nft does not minted from bridge)

✓ should fail if msg.sender is not the owner of nft

✓ should fail if msg.sender does not approve bridgeERC721 contract (38ms)

valid case

✓ should emit TokenDeposited event

#release()

valid case

✓ nft move to receiver address

✓ releasedDeposites information marked

✓ should emit TokenReleased event

setter function

✓ should fail if msg.sender does not have appropriate role (94ms)

✓ should fail if sanity check fail (64ms)

valid case

✓ parameter change properly

LiquidityMiningPoolsV1

LiquidityMiningPoolsV1 scenario test

✓ create LiquidityMiningPool by CrossChainBridgeERC20V1

✓ after creating pools, user staking (52ms)

✓ by staking, user earn rewardToken (222ms)

✓ user exit from pool (248ms)

✓ user transfer PoolsInterestBearingToken

RewardsPoolsV1

RewardsPoolsV1 scenario test

✓ create RewardsPools by CrossChainBridgeERC20V1

✓ after creating pools, user staking

✓ by staking, user earn rewardToken (202ms)

✓ user exit from pool (224ms)

✓ user transfer PoolsInterestBearingToken

End of Document