Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.08.16, the SlowMist security team received the ChainHop team's security audit application for ChainHop Iterative Audit, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit
		Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Audit Version:

https://github.com/chainhop-dex/chainhop-contracts

commit: 05b03bf6d094c0cff7062f10b2601ba43609cd5a

Fixed Version:

https://github.com/chainhop-dex/chainhop-contracts

commit: 51abab4c07165851e78f37039d97b25e65a3c305

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Forward hop logic flaw	Design Logic Audit	High	Ignored
N2	Safe Token Transfer	Design Logic Audit	Low	Fixed
N3	Token compatibility issues	Design Logic Audit	Low	Ignored

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

AnyswapAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
bridge	External	Payable	onlyMainContract
updateMainContract	External	Can Modify State	onlyOwner
setSupportedRouter	External	Can Modify State	onlyOwner

CBridgeAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
bridge	External	Payable	onlyMainContract
updateMainContract	External	Can Modify State	onlyOwner
executeMessageWithTransferRefund	External	Payable	onlyMessageBus

StargateAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
bridge	External	Payable	onlyMainContract
swap	Private	Can Modify State	-
updateMainContract	External	Can Modify State	onlyOwner
setSupportedRouter	External	Can Modify State	onlyOwner

StargateAdapter			
<Receive Ether>	External	Payable	-

CurveMetaPoolCodec			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	CurveTokenAddresses
decodeCalldata	External	-	-
encodeCalldataWithOverride	External	-	-

CurveSpecialMetaPoolCodec			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	CurveTokenAddresses
decodeCalldata	External	-	-
encodeCalldataWithOverride	External	-	-

CurveTokenAddresses			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setPoolTokens	External	Can Modify State	onlyOwner
_setPoolTokens	Private	Can Modify State	-

PlatypusRouter01Codec			
Function Name	Visibility	Mutability	Modifiers

PlatypusRouter01Codec			
decodeCalldata	External	-	-
encodeCalldataWithOverride	External	-	-

BridgeRegistry			
Function Name	Visibility	Mutability	Modifiers
setSupportedBridges	External	Can Modify State	onlyOwner

TransferSwapper			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Swapper FeeOperator SigVerifier
transferWithSwap	External	Payable	nonReentrant
_swapAndSend	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
executeMessageWithTransfer	External	Payable	onlyMessageBus nonReentrant
executeMessageWithTransferFallback	External	Payable	onlyMessageBus nonReentrant
executeMessageWithTransferRefund	External	Payable	onlyMessageBus nonReentrant
_refund	Private	Can Modify State	-
executeMessageWithTransferRefundFromAdapter	External	Can Modify State	nonReentrant

TransferSwapper			
_computeId	Private	-	-
_encodeRequestMessage	Internal	-	-
_encodeRequestMessage	Internal	-	-
_wrapBridgeOutToken	Private	Can Modify State	-
_sendToken	Private	Can Modify State	-
_verifyFee	Private	-	-
setNativeWrap	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-

4.3 Vulnerability Summary

[N1] [High] Forward hop logic flaw

Category: Design Logic Audit

Content

In the TransferSwapper contract, executeMessageWithTransfer is used for message execution and token transfer. If the dst chain needs another cbridge hop, it will be executed through `cBridge.bridge`, otherwise it will pay back to the executor. But before that if `_token` is nativeWrap , then part of `msg.value` will be converted to wrap token. Therefore, the native tokens in the contract are less than `msg.value` when performing `cBridge.bridge{value: msg.value}` and `_executor.call{value: msg.value}` operations.

Code location: contracts/TransferSwapper.sol

```
function executeMessageWithTransfer(
    address, // _sender
```

```
        address _token,
        uint256 _amount,
        uint64, // _srcChainId
        bytes memory _message,
        address _executor
    ) external payable override onlyMessageBus nonReentrant returns (ExecutionStatus)
{
        ...
        {
            _wrapBridgeOutToken(_token, _amount);
            address tokenOut = _token;
            ...
                forwardResp = cBridge.bridge{value: msg.value}(
                    f.dstChain,
                    m.receiver,
                    sumAmtOut,
                    tokenOut,
                    f.params,
                    requestMessage
                );
            } else {
                // msg.value is not used in this code branch, pay back to sender
                if (msg.value > 0) {
                    (bool sent, ) = _executor.call{value: msg.value}("");
                    require(sent, "send fail");
                }
                _sendToken(tokenOut, sumAmtOut, m.receiver, m.nativeOut);
            }
        }
        emit RequestDone(m.id, sumAmtOut, sumAmtFailed, _token, m.fee,
Types.RequestStatus.Succeeded, forwardResp);
        return ExecutionStatus.Success;
    }
```

Solution

If `_token` is a nativeWrap token, then `msg.value` should be used with caution.

Status

Ignored; After communicating with the project team, the project team stated that if to wrap, the NATIVE used to convert is from the ones sent by upstream (such as bridge) in advance, but not part of the msg.value, msg.value here is only used to pay for msg fee.

[N2] [Low] Safe Token Transfer

Category: Design Logic Audit

Content

In the executeMessageWithTransferRefundFromAdapter function of the TransferSwapper contract, it transfers tokens through the transferFrom function without checking the return value. If the token does not meet the EIP20 standard, there will be potential security risks.

The same is true for the bridge functions of the AnyswapAdapter, CBridgeAdapter and StargateAdapter contracts.

Code location:

contracts/TransferSwapper.sol

```
    function executeMessageWithTransferRefundFromAdapter(
        address _token,
        uint256 _amount,
        bytes calldata _message,
        address // _executor
    ) external nonReentrant returns (ExecutionStatus) {
        IERC20(_token).transferFrom(msg.sender, address(this), _amount);
        return _refund(_token, _amount, _message);
    }
```

contracts/bridges/AnyswapAdapter.sol

```
    function bridge(
        uint64 _dstChainId,
        address _receiver,
        uint256 _amount,
        address _token, // Note, here uses the address of the native
        bytes memory _bridgeParams,
        bytes memory //_requestMessage // Not used for now, as Anyswap messaging is
  not supported in this version
    ) external payable onlyMainContract returns (bytes memory bridgeResp) {
```

```
        ...
        IERC20(_token).transferFrom(msg.sender, address(this), _amount);
        ...
    }
```

contracts/bridges/CBridgeAdapter.sol

```
    function bridge(
        uint64 _dstChainId,
        address _receiver,
        uint256 _amount,
        address _token,
        bytes memory _bridgeParams,
        bytes memory _requestMessage
    ) external payable onlyMainContract returns (bytes memory bridgeResp) {
       ...
        IERC20(_token).transferFrom(msg.sender, address(this), _amount);
        ...
    }
```

contracts/bridges/StargateAdapter.sol

```
    function bridge(
        uint64 _dstChainId,
        address _receiver,
        uint256 _amount,
        address _token,
        bytes memory _bridgeParams,
        bytes memory //_requestMessage // Not used for now, as stargate messaging is
  not supported in this version
    ) external payable onlyMainContract returns (bytes memory bridgeResp) {
        ...
        IERC20(_token).transferFrom(msg.sender, address(this), _amount);
        ...
    }
```

Solution

It is recommended to use OpenZeppelin's SafeERC20 library for token transfers.

Status

Fixed

[N3] [Low] Token compatibility issues

Category: Design Logic Audit

Content

In the transferWithSwap function of the TransferSwapper contract, it will transfer the srcToken into this contract through the safeTransferFrom function. If srcToken is a deflationary token, the actual number of tokens received by the contract is less than the value of the amountIn parameter passed in by the user. This will result in cross-chain swap results that are not as expected. The same is true for the bridge functions of the AnyswapAdapter, CBridgeAdapter and StargateAdapter contracts.

Code location:

contracts/TransferSwapper.sol

```
    function transferWithSwap(
        Types.TransferDescription calldata _desc,
        ICodec.SwapDescription[] calldata _srcSwaps,
        ICodec.SwapDescription[] calldata _dstSwaps
    ) external payable nonReentrant {
        ...
        if (_desc.nativeIn) {
            require(srcToken == nativeWrap, "tkin no nativeWrap");
            require(msg.value >= amountIn, "insfcnt amt"); // insufficient amount
            IWETH(nativeWrap).deposit{value: amountIn}();
        } else {
            IERC20(srcToken).safeTransferFrom(msg.sender, address(this), amountIn);
        }

        _swapAndSend(srcToken, bridgeToken, amountIn, _desc, _srcSwaps, _dstSwaps,
codecs);
    }
```

contracts/bridges/AnyswapAdapter.sol

```
    function bridge(
        uint64 _dstChainId,
        address _receiver,
        uint256 _amount,
        address _token, // Note, here uses the address of the native
        bytes memory _bridgeParams,
        bytes memory //_requestMessage // Not used for now, as Anyswap messaging is
not supported in this version
    ) external payable onlyMainContract returns (bytes memory bridgeResp) {
        ...
        IERC20(_token).transferFrom(msg.sender, address(this), _amount);
        ...
    }
```

contracts/bridges/CBridgeAdapter.sol

```
    function bridge(
        uint64 _dstChainId,
        address _receiver,
        uint256 _amount,
        address _token,
        bytes memory _bridgeParams,
        bytes memory _requestMessage
    ) external payable onlyMainContract returns (bytes memory bridgeResp) {
       ...
        IERC20(_token).transferFrom(msg.sender, address(this), _amount
        ...
    }
```

contracts/bridges/StargateAdapter.sol

```
    function bridge(
        uint64 _dstChainId,
        address _receiver,
        uint256 _amount,
        address _token,
        bytes memory _bridgeParams,
        bytes memory //_requestMessage // Not used for now, as stargate messaging is
not supported in this version
    ) external payable onlyMainContract returns (bytes memory bridgeResp) {
```

```
        ...
        IERC20(_token).transferFrom(msg.sender, address(this), _amount);
        ...
    }
```

Solution

It is recommended to take the difference between the token balance of the contract before and after the user's transfer as the actual amount transferred by the user.

Status

Ignored; After communicating with the project team, the project team stated that the protocol does not support deflationary tokens.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002208220002	SlowMist Security Team	2022.08.16 - 2022.08.22	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high-risk and 2 low-risk vulnerabilities. 1 high-risk and 1 low-risk vulnerability were ignored; All other findings were fixed. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist