Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.03.16, the SlowMist security team received the ChainHop team's security audit application for ChainHop, 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: 6b6648239bc786b316f61e8817653d21f8014c5b

Fixed Version:

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

commit: 6a1487e5535314ee41b59f079e48f6d314b7bcd3

This audit does not include externally interacting contracts.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing event records	Others	Suggestion	Fixed
N2	Defective checking of return values	Design Logic Audit	Medium	Fixed
N3	Token Compatibility Issue	Design Logic Audit	Low	Confirmed
N4	Arbitrary external call risk	Design Logic Audit	Critical	Fixed
N5	Using unassigned parameters	Design Logic Audit	High	Fixed
N6	Problems with the same codecs	Design Logic Audit	Suggestion	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:

Codecs			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setCodec	Public	Can Modify State	onlyOwner
_setCodec	Private	Can Modify State	-
loadCodecs	Internal	-	-
getCodec	Internal	-	-

FeeOperator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
collectFee	External	Can Modify State	onlyFeeCollector
setFeeCollector	External	Can Modify State	onlyOwner

Multicall2			
Function Name	Visibility	Mutability	Modifiers

Multicall2			
aggregate	Public	Can Modify State	-
blockAndAggregate	Public	Can Modify State	-
getBlockHash	Public	-	-
getBlockNumber	Public	-	-
getCurrentBlockCoinbase	Public	-	-
getCurrentBlockDifficulty	Public	-	-
getCurrentBlockGasLimit	Public	-	-
getCurrentBlockTimestamp	Public	-	-
getEthBalance	Public	-	-
getLastBlockHash	Public	-	-
tryAggregate	Public	Can Modify State	-
tryBlockAndAggregate	Public	Can Modify State	-

SigVerifier			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setSigner	Public	Can Modify State	onlyOwner
verifySig	Internal	-	-

Swapper			
Function Name	Visibility	Mutability	Modifiers

Swapper			
sanitizeSwaps	Internal	-	-
executeSwaps	Internal	Can Modify State	-
executeSwapsWithOverride	Internal	Can Modify State	-
_redistributeAmountIn	Private	-	-
_executeSwapWithOverride	Internal	Can Modify State	-

TransferSwapper			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Codecs FeeOperator SigVerifier
transferWithSwap	External	Payable	-
_transfer	Private	Can Modify State	-
executeMessageWithTransfer	External	Payable	onlyMessageBus
executeMessageWithTransferFallback	External	Payable	onlyMessageBus
_sendToken	Private	Can Modify State	-
toHex	Public	-	-
_encodeRequestMessage	Private	-	-
_verifyFee	Private	-	-
setNativeWrap	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-

CurvePoolCodec			
Function Name	Visibility	Mutability	Modifiers
decodeCalldata	External	-	-
decodeReturnData	External	-	-
encodeCalldataWithOverride	External	-	-

UniswapV2SwapExactTokensForTokensCodec			
Function Name	Visibility	Mutability	Modifiers
decodeCalldata	External	-	-
decodeReturnData	External	-	-
encodeCalldataWithOverride	External	-	-

UniswapV3ExactInputSingleCodec			
Function Name	Visibility	Mutability	Modifiers
decodeCalldata	External	-	-
decodeReturnData	External	-	-
encodeCalldataWithOverride	External	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event records

Category: Others

Content

In the FeeOperator contract, the owner role can set the feeCollector parameter through the setFeeCollector function, but no event recording is performed.

In the SigVerifier contract, the owner role can set the signer parameter through the setSigner function, but no event recording is performed.

In the TransferSwapper contract, the owner role can set the nativeWrap parameter through the setNativeWrap function, but no event recording is performed.

Code location:

chainhop-contracts/contracts/FeeOperator.sol

```
    function setFeeCollector(address _feeCollector) external onlyOwner {
        feeCollector = _feeCollector;
    }
```

chainhop-contracts/contracts/SigVerifier.sol

```
    function setSigner(address _signer) public onlyOwner {
        signer = _signer;
    }
```

chainhop-contracts/contracts/TransferSwapper.sol

```
    function setNativeWrap(address _nativeWrap) external onlyOwner {
        nativeWrap = _nativeWrap;
    }
```

Solution

It is recommended to perform event logging on modification of sensitive parameters.

Status

Fixed

[N2] [Medium] Defective checking of return values

Category: Design Logic Audit

Content

In the executeSwapsWithOverride function of the Swapper contract, after the `_executeSwapWithOverride` operation, if `ok` is true but returns `(false, 0)`.

Code location: chainhop-contracts/contracts/Swapper.sol

```
    function executeSwapsWithOverride(
        ICodec.SwapDescription[] memory _swaps,
        uint256 _amountInOverride,
        ICodec[] memory _codecs // _codecs[i] is for _swaps[i]
    ) internal returns (bool ok, uint256 totalAmountOut) {
        uint256[] memory amountIns = _redistributeAmountIn(_swaps, _amountInOverride,
_codecs);
        for (uint256 i = 0; i < _swaps.length; i++) {
            uint256 amountOut;
            (ok, amountOut) = _executeSwapWithOverride(_codecs[i], _swaps[i],
amountIns[i]);
            if (ok) {
                return (false, 0);
            }
            totalAmountOut += amountOut;
        }
    }
```

Solution

If the design is unexpected, it is recommended to return `(false, 0)` when `ok` is false.

Status

Fixed

[N3] [Low] Token Compatibility Issue

Category: Design Logic Audit

Content

In the TransferSwapper contract, the transferWithSwap function is used to initiate cross-chain operations on the source chain. If tokenIn is not a native token, it will call safeTransferFrom to transfer the user-specified amountIn into this contract, and execute Swaps, _sendToken or _transfer operation. If tokenIn is a deflationary token, the number of tokens received by this contract will be less than the amountIn passed in by the user. This will cause problems with subsequent operations.

Code location: chainhop-contracts/contracts/TransferSwapper.sol

```
    function transferWithSwap(
        address _dstTransferSwapper,
        TransferDescription calldata _desc,
        ICodec.SwapDescription[] calldata _srcSwaps,
        ICodec.SwapDescription[] calldata _dstSwaps
    ) external payable {
        ...

        uint256 amountIn = _desc.amountIn;

        ...

        if (msg.value > 0) {
            // msg value > 0 automatically implies the sender wants to swap native
tokens
            require(msg.value >= amountIn, "insfcnt amt");
            IWETH(nativeWrap).deposit{value: msg.value}();
        } else {
            IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), amountIn);
        }
        ...
    }
```

Solution

It is recommended to obtain the token balance of this contract before and after the user's transferFrom as amountIn.

Status

Confirmed; After communicating with the project party, the project party stated that this protocol does not support deflationary tokens.

[N4] [Critical] Arbitrary external call risk

Category: Design Logic Audit

Content

In the TransferSwapper contract, the transferWithSwap and executeMessageWithTransfer functions can respectively call the executeSwaps function and the executeSwapsWithOverride function to perform the swap operation. This operation will be performed through a low-level call, but it does not check whether the dex and codecs in the _swaps parameter passed in by the user are expected. If the user passes in an unexpected value, it will cause any external call problems. For example, transfer the tokens of users who have approved this contract to others.

Code location: chainhop-contracts/contracts/TransferSwapper.sol

```
    function transferWithSwap(
        address _dstTransferSwapper,
        TransferDescription calldata _desc,
        ICodec.SwapDescription[] calldata _srcSwaps,
        ICodec.SwapDescription[] calldata _dstSwaps
    ) external payable {
        // a request needs to inccur a swap, a transfer, or both. otherwise it's a
nop and we revert early to save gas
        require(_srcSwaps.length != 0 || _desc.dstChainId != uint64(block.chainid),
"nop");
        require(_srcSwaps.length != 0 || (_desc.amountIn != 0 && _desc.tokenIn !=
address(0)), "nop");

        ...

        uint256 amountOut = amountIn;
        if (_srcSwaps.length != 0) {
            bool ok;
            (ok, amountOut) = executeSwaps(_srcSwaps, tokenIn, tokenOut, amountIn,
codecs);
            require(ok, "swap fail");
        }
```

```
        ...
    }

    function executeMessageWithTransfer(
        address, // _sender
        address _token,
        uint256 _amount,
        uint64, // _srcChainId
        bytes memory _message
    ) external payable override onlyMessageBus returns (bool ok) {
        ...

        if (m.swaps.length != 0) {
            // swap first before sending the token out to user
            (ok, dstAmount) = executeSwapsWithOverride(m.swaps, _amount, codecs);

        ...
    }

    function executeSwaps(
        ICodec.SwapDescription[] memory _swaps,
        address _tokenIn,
        address _tokenOut,
        uint256 _amountIn,
        ICodec[] memory _codecs // _codecs[i] is for _swaps[i]
    ) internal returns (bool ok, uint256 totalAmountOut) {
        uint256 balBefore = IERC20(_tokenOut).balanceOf(address(this));
        for (uint256 i = 0; i < _swaps.length; i++) {
            IERC20(_tokenIn).safeIncreaseAllowance(_swaps[i].dex, _amountIn);
            bytes memory res;
            (ok, res) = _swaps[i].dex.call(_swaps[i].data);

            ...
    }

    function executeSwapsWithOverride(
        ICodec.SwapDescription[] memory _swaps,
        uint256 _amountInOverride,
        ICodec[] memory _codecs // _codecs[i] is for _swaps[i]
    ) internal returns (bool ok, uint256 totalAmountOut) {
        uint256[] memory amountIns = _redistributeAmountIn(_swaps, _amountInOverride,
_codecs);
        // execute the swaps with adjusted amountIns
```

```
        for (uint256 i = 0; i < _swaps.length; i++) {
            uint256 amountOut;
            (ok, amountOut) = _executeSwapWithOverride(_codecs[i], _swaps[i],
amountIns[i]);
            if (ok) {
                return (false, 0);
            }
            totalAmountOut += amountOut;
        }
    }
```

Solution

It is recommended to check the parameters passed in by the user.

Status

Fixed

[N5] [High] Using unassigned parameters

Category: Design Logic Audit

Content

In the executeMessageWithTransfer function of the TransferSwapper contract, it defines codecs but does not assign a value to it, but passes in the executeSwapsWithOverride function for use.

Code location: chainhop-contracts/contracts/TransferSwapper.sol

```
    function executeMessageWithTransfer(
        address, // _sender
        address _token,
        uint256 _amount,
        uint64, // _srcChainId
        bytes memory _message
    ) external payable override onlyMessageBus returns (bool ok) {
        Request memory m = abi.decode((_message), (Request));
        address tokenOut;
        ICodec[] memory codecs;

        RequestStatus status;
```

```
        uint256 dstAmount = _amount;
        bool nativeOut = m.nativeOut;

        if (m.swaps.length != 0) {
            // swap first before sending the token out to user
            (ok, dstAmount) = executeSwapsWithOverride(m.swaps, _amount, codecs);

        ...
    }
```

Solution

It is recommended to assign parameters before use.

Status

Fixed

[N6] [Suggestion] Problems with the same codecs

Category: Design Logic Audit

Content

In the CodecRegistry contract, the owner can set the selector2codec mapping through the setCodec function. If there are different codecs but the selectors are the same, then when setting it again, the selector2codec set later will overwrite the previous one.

Code location: CodecRegistry.sol

```
    function setCodec(string calldata _funcSig, address _codec) public onlyOwner {
        bytes4 selector = bytes4(keccak256(bytes(_funcSig)));
        _setCodec(selector, _codec);
        emit CodecUpdated(selector, _codec);
    }

    function _setCodec(bytes4 _selector, address _codec) private {
        selector2codec[_selector] = ICodec(_codec);
        codecs.push(_codec);
    }
```

Solution

It is recommended to check if selector already exists when doing selector2codec setup.

Status

Ignored

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002203280003	SlowMist Security Team	2022.03.16 - 2022.03.28	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 1 high risk, 1 medium risk, 1 low risk, 2 suggestion vulnerabilities. And 1 low-risk vulnerability was confirmed and being fixed; 1 suggestion was 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