Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.10.14, the SlowMist security team received the TransitSwap team's security audit application for TransitSwap v4 Core, 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

Project address:

https://github.com/Transit-Finance/v4-core

commit: 80bae9f39815376035f540b1298360d49edb4ca5

Fixed Version

Project address:

https://github.com/Transit-Finance/v4-core

commit: aba27e0dd23bd82c207de46fc82667d91b3b6e73

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Authorization limit issue	Authority Control Vulnerability	Suggestion	Confirmed
N2	Security suggestion	Design Logic Audit	Suggestion	Confirmed
N3	Missing zero address check	Others	Suggestion	Confirmed
N4	Risk of excessive authority	Authority Control Vulnerability	Medium	Fixed
N5	Authority control issue	Authority Control Vulnerability	High	Fixed
N6	Uninitialized parameter	Others	Suggestion	Confirmed
N7	Unsafe external call	Others	Suggestion	Ignored

NO	Title	Category	Level	Status
N8	Missing Approve amount reset	Others	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code of this project has been open sourced on GitHub and the BSC mainnet.

GitHub

Project address: https://github.com/Transit-Finance/transit-core-v4

Commit: 2f34942b046ca6b31237a1b7c641a71dac7efd95

BSC Mainnet

TransitSwapRouter.sol: https://bscscan.com/address/0x9865EeBdD1cE65f45b6247AEEd2fA2252ECA7A08#code

TransitSwapFees: https://bscscan.com/address/0x1F6E41c47349634Fe261403A18F8515546f58826#code

TransitAllowed.sol: https://bscscan.com/address/0x28a03F7F7B8CA28500350c4A010F7D0393485395#code

TransitSwap.sol: https://bscscan.com/address/0xf7A2f863299C17dfA11CD8a14e7c7DCA92f315B9#code

TransitCross.sol: https://bscscan.com/address/0x8755773dc777B9F9B2E2c86402A03F099F823691#code

4.2 Visibility Description

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

TransitCross			
Function Name	Visibility	Mutability	Modifiers

TransitCross			
<Constructor>	Public	Can Modify State	Ownable
<Receive Ether>	External	Payable	-
transitRouter	Public	-	-
wrappedNative	Public	-	-
changeTransitRouter	Public	Can Modify State	onlyOwner
changeAllowed	Public	Can Modify State	onlyExecutor
cross	External	Payable	nonReentrant checkRouter
withdrawTokens	External	Can Modify State	onlyOwner

TransitSwap			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	-	-
<Receive Ether>	External	Payable	-
transitRouter	public	-	-
approves	public	-	-
changeTransitRouter	Public	Can Modify State	onlyOwner
_beforeSwap	Private	Can Modify State	-
swap	External	Payable	nonReentrant checkRouter
supportingFeeOn	External	Payable	nonReentrant checkRouter
_getAmountOut	Private	-	-

TransitSwap			
withdrawTokens	External	Can Modify State	onlyOwner

Ownable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
owner	Public	-	-
executor	Public	-	-
pendingOwner	Public	-	-
pendingExecutor	Public	-	-
_checkOwner	Internal	-	-
_checkExecutor	Internal	-	-
renounceOwnership	Public	Can Modify State	onlyOwner
transferOwnership	Public	Can Modify State	onlyOwner
transferExecutorship	Public	Can Modify State	onlyExecutor
_transferOwnership	Internal	Can Modify State	-
_transferExecutorship	Internal	Can Modify State	-
acceptOwnership	External	Can Modify State	-
acceptExecutorship	External	Can Modify State	-

ReentrancyGuard			
Function Name	Visibility	Mutability	Modifiers

ReentrancyGuard			
<Constructor>	Public	Can Modify State	-
_nonReentrantBefore	Private	Can Modify State	-
_nonReentrantAfter	Private	Can Modify State	-

TransitSwapFees			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
supportDiscount	Public	-	-
fees	Public	-	-
changeSupportDiscount	Public	Can Modify State	onlyOwner
setupTokens	Public	Can Modify State	onlyOwner
setupFees	Public	Can Modify State	onlyOwner
setupGradient	Public	Can Modify State	onlyOwner
getFeeRate	Public	-	-
withdrawTokens	External	Can Modify State	onlyOwner

TransitSwapRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
<Receive Ether>	External	Payable	-
transitSwap	External	-	-

TransitSwapRouter			
transitCross	External	-	-
transitFees	External	-	-
swapTypeMode	External	-	-
changeTransitSwap	External	Can Modify State	onlyExecutor
changeTransitCross	External	Can Modify State	onlyExecutor
changeTransitFees	External	Can Modify State	onlyExecutor
changeSwapTypeMode	External	Can Modify State	onlyOwner
_beforeSwap	Private	Can Modify State	-
_afterSwap	Private	Can Modify State	-
swap	External	Payable	nonReentrant whenNotPaused
_beforeCross	Private	Can Modify State	-
cross	External	Payable	nonReentrant whenNotPaused
_checkToken	Private	-	-
_emitTransit	Private	Can Modify State	-
withdrawTokens	External	Can Modify State	onlyOwner

Pausable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
paused	Public	-	-

Pausable			
_requireNotPaused	Internal	-	-
_requirePaused	Internal	-	-
_pause	Internal	Can Modify State	whenNotPaused
_unpause	Internal	Can Modify State	whenPaused

4.3 Vulnerability Summary

[N1] [Suggestion] Authorization limit issue

Category: Authority Control Vulnerability

Content

There are two roles Owner and Executor in the contract, and the permissions of the two roles are not clearly divided.

Solution

It is recommended to clearly divide the two permissions of parameter modification and fund management to facilitate permissions management.

Status

Confirmed; The project team uniformly uses the Executor role to manage parameter modification, deletes the owner role, and multi-signature will be used after the Executor role is deployed.

[N2] [Suggestion] Security suggestion

Category: Design Logic Audit

Content

Since the TransitSwapRouter contract will retain the user's authorization limit, it is recommended to allow the user to allocate on demand during the front-end authorization, and do not authorize the maximum value at one time to prevent the user's funds from being stolen.

Solution

It is recommended that the user manually set the authorization limit for each transaction.

Status

Confirmed

[N3] [Suggestion] Missing zero address check

Category: Others

Content

When modifying important addresses in the contract, it is not checked whether the incoming address is a zero address.

Code location: TransitCross.sol #L39-43

```
function changeTransitRouter(address newRouter) public onlyOwner {
    address oldRouter = _transit_router;
    _transit_router = newRouter;
    emit ChangeTransitRouter(oldRouter, newRouter);
}
```

Code location: TransitSwap.sol #L39-43

```
function changeTransitRouter(address newRouter) public onlyOwner {
    address oldRouter = _transit_router;
    _transit_router = newRouter;
    emit ChangeTransitRouter(oldRouter, newRouter);
}
```

Code location: TransitSwapRouter.sol #L73-89

```
function changeTransitSwap(address newTransit) external onlyExecutor {
    address oldTransit = _transit_swap;
    _transit_swap = newTransit;
    emit ChangeTransitSwap(oldTransit, newTransit);
}
```

```
    function changeTransitCross(address newTransit) external onlyExecutor {
        address oldTransit = _transit_cross;
        _transit_cross = newTransit;
        emit ChangeTransitCross(oldTransit, newTransit);
    }

    function changeTransitFees(address newTransitFees) external onlyExecutor {
        address oldTransitFees = _transit_fees;
        _transit_fees = newTransitFees;
        emit ChangeTransitFees(oldTransitFees, newTransitFees);
    }
```

Solution

It is recommended to add a zero address check.

Status

Confirmed

[N4] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

The Owner has the risk of over-authorization, and this role can withdraw the tokens in the contract to any address.

Code location: TransitCross.sol #L84-96

```
    function withdrawTokens(address[] memory tokens, address recipient) external
onlyOwner {
        for(uint index; index < tokens.length; index++) {
            uint amount;
            if (TransferHelper.isETH(tokens[index])) {
                amount = address(this).balance;
                TransferHelper.safeTransferETH(recipient, amount);
            } else {
                amount = IERC20(tokens[index]).balanceOf(address(this));
                TransferHelper.safeTransfer(tokens[index], recipient, amount);
            }
            emit Withdraw(tokens[index], msg.sender, recipient, amount);
```

```
        }
    }
```

Code location: TransitSwap.sol #L134-146

```
    function withdrawTokens(address[] memory tokens, address recipient) external
onlyOwner {
        for(uint index; index < tokens.length; index++) {
            uint amount;
            if (TransferHelper.isETH(tokens[index])) {
                amount = address(this).balance;
                TransferHelper.safeTransferETH(recipient, amount);
            } else {
                amount = IERC20(tokens[index]).balanceOf(address(this));
                TransferHelper.safeTransfer(tokens[index], recipient, amount);
            }
            emit Withdraw(tokens[index], msg.sender, recipient, amount);
        }
    }
```

Code location: TransitSwapFees.sol #L95-107

```
    function withdrawTokens(address[] memory tokens, address recipient) external
onlyOwner {
        for(uint index; index < tokens.length; index++) {
            uint amount;
            if(TransferHelper.isETH(tokens[index])) {
                amount = address(this).balance;
                TransferHelper.safeTransferETH(recipient, amount);
            } else {
                amount = IERC20(tokens[index]).balanceOf(address(this));
                TransferHelper.safeTransfer(tokens[index], recipient, amount);
            }
            emit Withdraw(tokens[index], msg.sender, recipient, amount);
        }
    }
```

Code location: TransitSwapRouter.sol #L272-284

```
    function withdrawTokens(address[] memory tokens, address recipient) external
onlyOwner {
        for(uint index; index < tokens.length; index++) {
            uint amount;
            if(TransferHelper.isETH(tokens[index])) {
                amount = address(this).balance;
                TransferHelper.safeTransferETH(recipient, amount);
            } else {
                amount = IERC20(tokens[index]).balanceOf(address(this));
                TransferHelper.safeTransfer(tokens[index], recipient, amount);
            }
            emit Withdraw(tokens[index], msg.sender, recipient, amount);
        }
    }
```

Owner has the risk of over-authorization. This role can modify swapType, feeRate, gradientThreshold and gradientDiscount parameters through the setupFees function and setupGradient function, and there is no limit to the value range. These parameters will affect the calculation result of the handling fee.

Code location: TransitSwapFees.sol #L51-64

```
    function setupFees(uint256[] memory swapType, uint256[] memory feeRate, string[]
memory channel) public onlyOwner {
        require(swapType.length == feeRate.length, "invalid data");
        require(swapType.length == channel.length, "invalid data");
        for(uint256 index; index < swapType.length; index++) {
            _fees[swapType[index]][channel[index]] = feeRate[index];
        }
        emit SetupFees(swapType, feeRate, channel);
    }

    function setupGradient(uint256[] memory gradientThreshold, uint256[] memory
gradientDiscount) public onlyOwner {
        _gradient_threshold = gradientThreshold;
        _gradient_discount = gradientDiscount;
        emit SetupGradient(gradientThreshold, gradientDiscount);
    }
```

The TransitSwapRouter contract will obtain the user's authorization limit. The Executor role has the right to modify the addresses of the _transit_swap, _transit_cross and _transit_fees contracts. If the address is a malicious address, the user's assets will be stolen.

Code location: TransitSwapRouter.sol #L59-75

```
    function changeTransitSwap(address newTransit) external onlyExecutor {
        address oldTransit = _transit_swap;
        _transit_swap = newTransit;
        emit ChangeTransitSwap(oldTransit, newTransit);
    }

    function changeTransitCross(address newTransit) external onlyExecutor {
        address oldTransit = _transit_cross;
        _transit_cross = newTransit;
        emit ChangeTransitCross(oldTransit, newTransit);
    }

    function changeTransitFees(address newTransitFees) external onlyExecutor {
        address oldTransitFees = _transit_fees;
        _transit_fees = newTransitFees;
        emit ChangeTransitFees(oldTransitFees, newTransitFees);
    }
```

Solution

It is recommended to transfer the Owner and Executor role to TimeLock contract governance, and at least multi-signature management should be used. When modifying the handling fee calculation parameter, its value range should be limited.

Status

Fixed; The project team has deprecated the Owner permission and handed over all permissions to the Executor role for management. After deployment, the authority of the Executor role has been transferred to the multi-signature contract. The following are the authority transfer transactions of each main network:

BSC(https://bscscan.com)

TXHash: 0xedc4c69239262b17a5f1c9391f37d1a7fdbaff762e8174eb79bd0039ba5180a3

ETH(https://cn.etherscan.com)

TXHsh: 0x3da3b0cad4abafda894d7e082614f82e601bf4d65340dc24ce56a877e6665846

Polygon(https://polygonscan.com)

TXHash: 0xc6e93c4d4b9585c9c64ee36ad7b37e59abec4d734aac0dec1c50e78e5253c37b

AVAX(https://snowtrace.io)

TXHash: 0xe064ca1d5f5147ef1703dbadc558754a1034a49cb61d15c7b42999af36272263

ARB(https://arbiscan.io)

TXHash: 0x56bc22418a660f1052019bae57fb1e2006d170a20e083d7d1db88144fbca2241

[N5] [High] Authority control issue

Category: Authority Control Vulnerability

Content

The contract adopts a completely open calling logic. There is an operation to authorize the approveAddress address in the _beforeSwap function. The calling path of this function is TransitSwapRouter.swap() -> TransitSwap.swap() -> TransitSwap._beforeSwap(). The calldata parameter is also passed in when calling the top-level function TransitSwapRouter.swap(). The code does not check whether the approveAddress is legal. If a malicious approveAddress is passed in, the contract will be incorrectly authorized and the tokens in the contract will be lost.

Code location: TransitSwap.sol #L45-59

```
    function _beforeSwap(address srcToken, address dstToken, address approveAddress)
private returns (uint256 balance) {
        if (TransferHelper.isETH(dstToken)) {
            balance = address(this).balance;
        } else {
            balance = IERC20(dstToken).balanceOf(address(this));
        }

        if (!TransferHelper.isETH(srcToken)) {
            bool isApproved = _approves[srcToken][approveAddress];
            if (!isApproved) {
                TransferHelper.safeApprove(srcToken, approveAddress, 2**256-1);
```

```
                _approves[srcToken][approveAddress] = true;
            }
        }
    }
```

Solution

It is recommended to add a whitelist that restricts external call addresses to prevent malicious external calls.

Status

Fixed; The project team added a call whitelist in the TransitAllowed contract management contract to limit open external calls.

[N6] [Suggestion] Uninitialized parameter

Category: Others

Content

The swapAmount parameter is declared in the contract but not initialized.

Code location: TransitCross.sol #L64-87

```
    function cross(address srcToken, TransitStructs.CrossDescription memory
crossDesc) internal {

        require(_cross_allowed[crossDesc.caller], "TransitSwap: caller not allowed");
        uint swapAmount;

        if (TransferHelper.isETH(srcToken)) {
            require(msg.value >= crossDesc.amount, "TransitSwap: Invalid msg.value");
            swapAmount = msg.value;

            if (crossDesc.needWrapped) {
                TransferHelper.safeDeposit(_wrapped, crossDesc.amount);
                TransferHelper.safeApprove(_wrapped, crossDesc.caller, swapAmount);
                swapAmount = 0;
            }
        } else {
            require(IERC20(srcToken).balanceOf(address(this)) >= crossDesc.amount,
"TransitSwap: Invalid amount");
            TransferHelper.safeApprove(srcToken, crossDesc.caller, crossDesc.amount);
```

```
        }

        (bool success, bytes memory result) = crossDesc.caller.call{value:swapAmount}
(crossDesc.calls);
        if (!success) {
            revert(RevertReasonParser.parse(result, ""));
        }
    }
```

Solution

It is recommended to initialize the swapAmount parameter.

Status

Confirmed

[N7] [Suggestion] Unsafe external call

Category: Others

Content

The path and pair parameters in the supportingFeeOn function are controllable. If an attacker passes in malicious path and pair parameters, it may cause unexpected errors.

Code location: TransitSwap.sol #L62-72

```
    function callbytes(TransitStructs.CallbytesDescription calldata desc) external
payable nonReentrant checkRouter {
        if (desc.flag == uint8(TransitStructs.Flag.aggregate)) {
            TransitStructs.AggregateDescription memory aggregateDesc =
TransitStructs.decodeAggregateDesc(desc.calldatas);
            swap(desc.srcToken, aggregateDesc);
        } else if (desc.flag == uint8(TransitStructs.Flag.swap)) {
            TransitStructs.SwapDescription memory swapDesc =
TransitStructs.decodeSwapDesc(desc.calldatas);
            supportingFeeOn(swapDesc);
        } else {
            revert("TransitSwap: invalid flag");
        }
    }
```

Code location: TransitSwap.sol #L104-130

```
    function supportingFeeOn(TransitStructs.SwapDescription memory desc) internal {
        require(desc.deadline >= block.timestamp, "TransitSwap: expired");
        require(desc.paths.length == desc.pairs.length, "TransitSwap: invalid
calldatas");
        for (uint i; i < desc.paths.length; i++) {
            address[] memory path = desc.paths[i];
            address[] memory pair = desc.pairs[i];
            uint256 fee = desc.fees[i];
            for (uint256 index; index < path.length - 1; index++) {
                (address input, address output) = (path[index], path[index + 1]);
                (address token0,) = input < output ? (input, output) : (output,
input);
                IUniswapV2Pair pairAddress = IUniswapV2Pair(pair[index]);
                uint amountInput;
                uint amountOutput;
                {
                    // scope to avoid stack too deep errors
                    (uint reserve0, uint reserve1,) = pairAddress.getReserves();
                    (uint reserveInput, uint reserveOutput) = input == token0 ?
(reserve0, reserve1) : (reserve1, reserve0);
                    amountInput =
IERC20(input).balanceOf(address(pairAddress)).sub(reserveInput);
                    //getAmountOut
                    amountOutput = _getAmountOut(amountInput, reserveInput,
reserveOutput, fee);
                }
                (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0),
amountOutput) : (amountOutput, uint(0));
                address to = index < path.length - 2 ? pair[index + 1] :
desc.receiver;
                pairAddress.swap(amount0Out, amount1Out, to, new bytes(0));
            }
        }
    }
```

Solution

It is recommended to use a whitelist to restrict incoming paths and pairs.

Status

Ignored; After communication, we learned that the problem will not affect the user's funds, so it will not be dealt with.

[N8] [Suggestion] Missing Approve amount reset

Category: Others

Content

The _beforeSwap function in the TransitSwap contract will set the authorization limit to the maximum value when accessing the token for the first time, but the function to remake the authorization is not found in the contract. When the authorization limit is used up, it will not be able to remake and the token cannot be used.

Solution

It is recommended to add the reset approve amount function to reset the approve amount.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002210200001	SlowMist Security Team	2022.10.14 - 2022.10.20	Passed

Summary conclusion: The SlowMist security team used a manual and SlowMist team's analysis tool for auditing the project, during the audit work we found 1 high risk, 1 medium risk, and 6 suggestion vulnerabilities. 1 high risk and 1 medium risk have been fixed; 1 suggestion vulnerability was ignored; All other findings were confirmed. The code has been 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