SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2024.08.15, the SlowMist security team received the KayakFi team's security audit application for Kayak-UniswapV3-Contract, 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.
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
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

Serial Number	Audit Class	Audit Subclass
7	Security Design 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

This is the DEX part of KayaFi, which mainly includes Core, Periphery and SwapRouter modules. Core and Periphery modules are forks of Uniswap v3

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing zero address check	Others	Suggestion	Fixed

NO	Title	Category	Level	Status
N2	The identity of msg.sender is not verified	Authority Control Vulnerability Audit	Low	Fixed
N3	Unchecked return value	Others	Low	Fixed
N4	Redundant code	Others	Suggestion	Fixed
N5	Insufficient ETH balance causes the function to be unavailable	Design Logic Audit	Medium	Fixed
N6	External dependency changes may cause logic failure	Others	Information	Acknowledged
N7	Returns incorrect swap result	Design Logic Audit	Low	Fixed
N8	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Acknowledged
N9	Redundant authorization operations	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/KayakFi/Kayak-UniswapV3-Contract

Initial audit version: 6b8117f2a1b19911a067899c8d67a01ebf897601

Final aduit version: e0299211edfdb56acb0604534cb5313f002dbcca

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:

NoDelegateCall			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
checkNotDelegateCall	Private	-	-

UniswapV3Factory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
createPool	External	Can Modify State	noDelegateCall
setOwner	External	Can Modify State	-
enableFeeAmount	Public	Can Modify State	-

UniswapV3PoolDeployer			
Function Name	Visibility	Mutability	Modifiers
deploy	Internal	Can Modify State	-

UniswapV3Pool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
checkTicks	Private	-	-
_blockTimestamp	Internal	-	-
balance0	Private	-	-
balance1	Private	-	-
snapshotCumulativesInside	External	-	noDelegateCall

UniswapV3Pool			
observe	External	-	noDelegateCall
increaseObservationCardinalityNext	External	Can Modify State	lock noDelegateCall
initialize	External	Can Modify State	-
_modifyPosition	Private	Can Modify State	noDelegateCall
_updatePosition	Private	Can Modify State	-
mint	External	Can Modify State	lock
collect	External	Can Modify State	lock
burn	External	Can Modify State	lock
swap	External	Can Modify State	noDelegateCall
flash	External	Can Modify State	lock noDelegateCall
setFeeProtocol	External	Can Modify State	lock onlyFactoryOwner
collectProtocol	External	Can Modify State	lock onlyFactoryOwner

NonfungiblePositionManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721Permit PeripheryImmutableState
positions	External	-	-
cachePoolKey	Private	Can Modify State	-
mint	External	Payable	checkDeadline
tokenURI	Public	-	-
baseURI	Public	-	-
increaseLiquidity	External	Payable	checkDeadline
decreaseLiquidity	External	Payable	isAuthorizedForToken checkDeadline

NonfungiblePositionManager			
collect	External	Payable	isAuthorizedForToken
burn	External	Payable	isAuthorizedForToken
_getAndIncrementNonce	Internal	Can Modify State	-
getApproved	Public	-	-
_approve	Internal	Can Modify State	-

NonfungibleTokenPositionDescriptor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
nativeCurrencyLabel	Public	-	-
tokenURI	External	-	-
flipRatio	Public	-	-
tokenRatioPriority	Public	-	-

SwapRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	PeripheryImmutableState
getPool	Private	-	-
uniswapV3SwapCallback	External	Can Modify State	-
exactInputInternal	Private	Can Modify State	-
exactInputSingle	External	Payable	checkDeadline
exactInput	External	Payable	checkDeadline
exactOutputInternal	Private	Can Modify State	-

SwapRouter			
exactOutputSingle	External	Payable	checkDeadline
exactOutput	External	Payable	checkDeadline

KayakSwapQuoter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Fallback>	External	Can Modify State	-
<Receive Ether>	External	Payable	-
swapMultiReturn	External	Can Modify State	-
swapReturn	External	Can Modify State	-
getReturn	Public	Can Modify State	-
getReturnStableSwap	Public	-	-
getReturnUniswapV3	Public	Can Modify State	-
parseRevertReason	Private	-	-
uniswapV3SwapCallback	External	-	-

KayakSwapRouter			
Function Name	Visibility	Mutability	Modifiers
<Fallback>	External	Can Modify State	-
<Receive Ether>	External	Payable	-
<Constructor>	Public	Can Modify State	-
swapMulti	Public	Payable	nonReentrant
swap	Public	Payable	nonReentrant

KayakSwapRouter			
_swapOnStableSwap	Internal	Can Modify State	-
_swapOnV3ExactIn	Internal	Can Modify State	-
uniswapV3SwapCallback	External	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing zero address check

Category: Others

Content

1.In the KayakSwapRouter contract, the `constructor` function lacks a zero address check for the `_WETH` parameter.

- contracts/swaprouter/KayakSwapRouter.sol#L67-L69

```
    constructor(address _WETH) {
        weth = IWETH(_WETH);
    }
```

2.In the KayakSwapRouter contract, the `swap` function lacks a zero address check for the `params.pool` parameter.

- contracts/swaprouter/KayakSwapRouter.sol#L113-L141

~~~~
    function swap(SwapParams calldata params) public payable nonReentrant returns
 (uint256 returnAmount) {
        ```
        if (params.flag) {
            _swapOnStableSwap(params.srcToken, params.dstToken, params.pool,
receivedAmount);
        } else {
            _swapOnV3ExactIn(params.srcToken, params.dstToken, params.pool,
receivedAmount);
        }
        ```
    }
~~~~

3.In the KayakSwapQuoter contract, the `constructor` function lacks a zero address check for the `_WETH` parameter.

- contracts/swaprouter/KayakSwapQuoter.sol#L34-L36

~~~
constructor(address _WETH) {
    weth = IWETH(_WETH);
}
~~~

4.In the KayakSwapQuoter contract, the `getReturnStableSwap` function and `getReturnUniswapV3` function lack a zero address check for the `pool` parameter.

- contracts/swaprouter/KayakSwapQuoter.sol#L87-L103, L105-L130

~~~
function getReturnStableSwap(
    address srcToken,
    address dstToken,
    address pool,
    uint256 amount
) public view returns (uint256 returnAmount) {
    ```
    uint256 n_coins = IStableSwap(pool).N_COINS();
    ```
}

function getReturnUniswapV3(
    address srcToken,
    address dstToken,
    address pool,
    uint256 amount
) public returns (uint256 returnAmount) {
    ```
        IUniswapV3Pool(pool).swap(
    ```
        )
    ```
}
~~~

Solution

It is recommended to add a zero address check to the address parameter.

Status

Fixed

[N2] [Low] The identity of msg.sender is not verified

Category: Authority Control Vulnerability Audit

Content

In the KayakSwapRouter contract, the `uniswapV3SwapCallback` function does not verify whether `msg.sender` is a valid Uniswap V3 Pool. When there are assets in the contract, the attacker can construct malicious parameters and transfer any assets in the contract through the `uniTransfer` function called in the `uniswapV3SwapCallback` function.

- contracts/swaprouter/KayakSwapRouter.sol#L187-L197

```
    function uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta, bytes calldata _data) external {
        require(amount0Delta > 0 || amount1Delta > 0); // swaps entirely within 0-liquidity regions are not supported
        SwapCallbackData memory data = abi.decode(_data, (SwapCallbackData));
        (address tokenIn, address tokenOut, , ) = data.path.decodeFirstPool();

        (, uint256 amountToPay) = amount0Delta > 0
            ? (tokenIn < tokenOut, uint256(amount0Delta))
            : (tokenOut < tokenIn, uint256(amount1Delta));

        IERC20(tokenIn).uniTransfer(msg.sender, amountToPay);
    }
```

Solution

It is recommended to verify that msg.sender is a valid Uniswap V3 Pool.

Status

Fixed

[N3] [Low] Unchecked return value

Category: Others

Content

In the UniERC20 library, the `uniApprove` function did not check the return value when calling the `approve` function.

- contracts/swaprouter/libraries/UniERC20.sol#L68-L72

```
function uniApprove(IERC20 token, address to, uint256 amount) internal {
    if (isETH(token)) return;

    token.approve(to, amount);
}
```

Solution

It is recommended to check the return value of the called function.

Status

Fixed

[N4] [Suggestion] Redundant code

Category: Others

Content

In the KayakSwapQuoter contract, `amountOutCached` is not used.

- contracts/swaprouter/KayakSwapQuoter.sol#L22

```
uint256 private amountOutCached;
```

Solution

It is recommended to remove redundant code.

Status

Fixed

[N5] [Medium] Insufficient ETH balance causes the function to be unavailable

Category: Design Logic Audit

Content

In the KayakSwapQuoter contract, the purpose of the `getReturnUniswapV3` function is to return the number of tokens that can be exchanged in the UniswapV3 pool, so there is no need to call the `weth.deposit` function to exchange WETH. Calling the `weth.deposit` function will fail because the contract does not have enough ETH balance and subsequent operations cannot be performed, which will make the function unusable.

- contracts/swaprouter/KayakSwapQuoter.sol

```
    function getReturnUniswapV3(
        address srcToken,
        address dstToken,
        address pool,
        uint256 amount
    ) public returns (uint256 returnAmount) {
        if (IERC20(srcToken).isETH()) {
            weth.deposit{ value: amount }();
        }

        address srcTokenReal = IERC20(srcToken).isETH() ? address(weth) : srcToken;
        address dstTokenReal = IERC20(dstToken).isETH() ? address(weth) : dstToken;
        bool zeroForOne = srcTokenReal < dstTokenReal;

        try
            IUniswapV3Pool(pool).swap(
                address(this), // address(0) might cause issues with some tokens
                zeroForOne,
                amount.toInt256(),
                zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO -
1,
                abi.encodePacked(srcTokenReal, dstTokenReal, pool, false)
            )
        {} catch (bytes memory reason) {
            return parseRevertReason(reason);
        }
    }
```

Solution

It is recommended to remove the call to the weth.deposit function and directly execute subsequent operations.

Status

Fixed

[N6] [Information] External dependency changes may cause logic failure

Category: Others

Content

In the KayakSwapQuoter contract, the `parseRevertReason` function can parse the error information returned by UniswapV3Pool. If the Uniswap V3 contract interface or error return format changes, it may cause errors in error message parsing.

- contracts/swaprouter/KayakSwapQuoter.sol#L133-L142

```
function parseRevertReason(bytes memory reason) private pure returns (uint256) {
    if (reason.length != 32) {
        if (reason.length < 68) revert("Unexpected error");
        assembly {
            reason := add(reason, 0x04)
        }
        revert(abi.decode(reason, (string)));
    }
    return abi.decode(reason, (uint256));
}
```

Solution

It is recommended that the Uniswap V3 contract interface or error return format be changed, and the parsing logic also needs to be modified synchronously.

Status

Acknowledged; The project stated that they will synchronously modify the parsing logic when the return format of swap contracts is changed.

[N7] [Low] Returns incorrect swap result

Category: Design Logic Audit

Content

In the KayakSwapQuoter contract, the `getReturnStableSwap` function and `getReturnUniswapV3` function do not verify whether the `pool` contract is the correct address. If the user passes in a malicious contract address, an incorrect exchange result may be returned.

- contracts/swaprouter/KayakSwapQuoter.sol#L87-L103, L105-L130

```
    function getReturnStableSwap(
        address srcToken,
        address dstToken,
        address pool,
        uint256 amount
    ) public view returns (uint256 returnAmount) {
        address[] memory tokens = new address[](3);
        uint256 n_coins = IStableSwap(pool).N_COINS();
        tokens[0] = IStableSwap(pool).coins(uint256(0));
        tokens[1] = IStableSwap(pool).coins(uint256(1));
        if (n_coins == 3) {
            tokens[2] = IStableSwap(pool).coins(uint256(2));
        }
        uint256 i = (srcToken == tokens[0] ? 1 : 0) + (srcToken == tokens[1] ? 2 : 0)
+ (srcToken == tokens[2] ? 3 : 0);
        uint256 j = (dstToken == tokens[0] ? 1 : 0) + (dstToken == tokens[1] ? 2 : 0)
+ (dstToken == tokens[2] ? 3 : 0);
        return IStableSwap(pool).get_dy(i - 1, j - 1, amount);
    }

    function getReturnUniswapV3(
        address srcToken,
        address dstToken,
        address pool,
        uint256 amount
    ) public returns (uint256 returnAmount) {
        if (IERC20(srcToken).isETH()) {
            weth.deposit{ value: amount }();
        }

        address srcTokenReal = IERC20(srcToken).isETH() ? address(weth) : srcToken;
        address dstTokenReal = IERC20(dstToken).isETH() ? address(weth) : dstToken;
        bool zeroForOne = srcTokenReal < dstTokenReal;

        try
            IUniswapV3Pool(pool).swap(
                address(this), // address(0) might cause issues with some tokens
                zeroForOne,
                amount.toInt256(),
                zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO -
1,
                abi.encodePacked(srcTokenReal, dstTokenReal, pool, false)
            )
        {} catch (bytes memory reason) {
            return parseRevertReason(reason);
        }
    }
```

Solution

It is recommended to verify whether the passed in pool contract is created by the corresponding factory contract.

Status

Fixed

[N8] [Low] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the UniswapV3Factory contract, the `owner` role can transfer `owner` permissions and set `feeAmountTickSpacing` mapping.

- contracts/core/UniswapV3Factory.sol#L54-L58, L61-L72

```
    function setOwner
    function enableFeeAmount
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged; The project team stated that they will use a multi-signature contract to manage the owner role.

[N9] [Low] Redundant authorization operations

Category: Design Logic Audit

Content

In the KayakSwapRouter contract, the `_swapOnV3ExactIn` function calls the `uniApprove` function to authorize the pool contract, but in the `uniswapV3SwapCallback` function, the `uniTransfer` function is called to pay the `tokenIn` token, and the `uniTransferFrom` function is not used, so the authorized amount will continue to exist and accumulate. If the pool contract is a malicious contract, this authorization operation may cause the assets in the KayakSwapRouter contract to be transferred away.

- contracts/swaprouter/KayakSwapRouter.sol#L162-L185, L187-L197

```
    function _swapOnV3ExactIn(address srcToken, address dstToken, address pool,
uint256 amount) internal {
        ```
        IERC20(srcTokenReal).uniApprove(payable(pool), amount);
        ```
    }

    function uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta, bytes
calldata _data) external {
        ```
        IERC20(tokenIn).uniTransfer(msg.sender, amountToPay);
    }
```

Solution

It is recommended to remove the code that calls the uniApprove function in the _swapOnV3ExactIn function or modify the uniswapV3SwapCallback function to call the uniTransferFrom function to pay the token.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002408200001	SlowMist Security Team	2024.08.15 - 2024.08.20	Low Risk

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 medium risk, 5 low risk, 2 suggestion, 1 Information.
```

# 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.

SLOWMIST

**Official Website**

www.slowmist.com

**E-mail**

team@slowmist.com

**Twitter**

@SlowMist_Team

**Github**

https://github.com/slowmist