SLOWMIST

Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.11.27, the SlowMist security team received the LiquiX team's security audit application for LiquiX, 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 a yield protocol, including Vault parts.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	No slippage protection created	Design Logic Audit	High	Fixed
N2	Withdrawal amount is inaccurate	Design Logic Audit	Low	Fixed

NO	Title	Category	Level	Status
N3	WETH tokens are not unwrapped	Design Logic Audit	High	Fixed
N4	Missing event record	Others	Suggestion	Acknowledged
N5	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N6	Redundant code	Others	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Initial audit sha256(contract.zip) = 5292f6faa57f86efc2effd5016d7ce9e4b72bdd40bebc5d3758c4c79d0d1b567

Final aduit sha256(contract.zip) = 4fb3ad4004875c7b4a3fe1893bbc836323d22927756e581b4dc80812aa5d06fd

```
.
├── Vault.sol
├── interfaces
|   ├── IERC20.sol
|   ├── INonfungiblePositionManager.sol
|   ├── IPool.sol
|   ├── IRewardTracker.sol
|   ├── ISwapRouter02.sol
|   └── IWETH.sol
└── libs
├── AaveHelper.sol
├── LiquidityHelper.sol
├── SwapHelper.sol
├── TransferHelper.sol
└── VaultStructInfo.sol
```

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:

Vault			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	onlyOwner
getVaultName	Public	-	-
updateVaultName	External	Can Modify State	onlyOwner
onERC721Received	External	-	-
swapInputETHForToken	External	Can Modify State	dispatcherCheck allowListCheck
swapInputForErc20Token	External	Can Modify State	dispatcherCheck allowListCheck
swapInputTokenToETH	External	Can Modify State	dispatcherCheck
mintPosition	Public	Can Modify State	dispatcherCheck allowListCheck allowListCheck
mintPositions	External	Can Modify State	dispatcherCheck
increaseLiquidity	External	Can Modify State	dispatcherCheck
removeAllPositionById	Public	Can Modify State	dispatcherCheck
removeAllPositionByIds	External	Can Modify State	dispatcherCheck

Vault			
removeLpInfoByTokenIds	External	Can Modify State	dispatcherCheck
collectAllFees	External	Can Modify State	dispatcherCheck
burnNFT	External	Can Modify State	dispatcherCheck
collectAllFeesInner	Internal	Can Modify State	-
depositAllToAave	External	Can Modify State	dispatcherCheck
withdrawAllFromAave	External	Can Modify State	dispatcherCheck
withdrawFromAaveForTrading	Internal	Can Modify State	-
withdrawAllFromAave	Internal	Can Modify State	-
depositToAave	Internal	Can Modify State	-
setDispatcher	External	Can Modify State	onlyOwner
setSwapAllowList	External	Can Modify State	onlyOwner
updateTradingFee	External	Can Modify State	onlyDispatcherCheck
setAutoStake	External	Can Modify State	onlyOwner
claimRewards	External	Can Modify State	onlyOwner
getPositionIds	External	-	-
getTokenIdByCustomerId	Public	-	-
queryRemovedLpInfo	Public	-	-
getAllowTokenList	Public	-	-
balanceOf	Public	-	-

Vault			
isAutoStake	Public	-	-
<Receive Ether>	External	Payable	-
withdrawErc721NFT	External	Can Modify State	onlyOwner
withdrawTokens	External	Can Modify State	onlyOwner
withdrawETH	External	Can Modify State	onlyOwner
deposit	External	Can Modify State	onlyOwner
depositEthToWeth	External	Payable	onlyOwner
depositGasToDispatcher	External	Can Modify State	dispatcherCheck

4.3 Vulnerability Summary

[N1] [High] No slippage protection created

Category: Design Logic Audit

Content

When the `mintPosition` function, `mintPositions` function, `increaseLiquidity` function, `removeAllPositionById` function, and `removeAllPositionByIds` function of the Vault contract call the `mintNewPosition` function, `increaseLiquidityCurrentRange` function, and `removeAllPositionById` function in the LiquidityHelper library respectively, the `amount0Min` and `amoun1Min` parameters passed in are set to 0, and no slippage is created. protections, are vulnerable to front-running attacks designed to execute calls at inaccurate prices.

- contracts/libs/LiquidityHelper.sol#L32-63,L65-L77,L92-L100

```
    function mintNewPosition(PositionMap storage positionMap, CreateLpObject memory
createLpObj, INonfungiblePositionManager nonfungiblePositionManager, mapping(address
=> bool) storage approveMap) internal returns (uint256 tokenId, uint256 amount0,
uint256 amount1) {
        if(!approveMap[createLpObj.token0]) {
            TransferHelper.safeApprove(createLpObj.token0,
address(nonfungiblePositionManager), type(uint256).max);
            approveMap[createLpObj.token0] = true;
        }
        if(!approveMap[createLpObj.token1]) {
            TransferHelper.safeApprove(createLpObj.token1,
address(nonfungiblePositionManager), type(uint256).max);
            approveMap[createLpObj.token1] = true;
        }
        INonfungiblePositionManager.MintParams memory params =
INonfungiblePositionManager.MintParams({
                token0: createLpObj.token0,
                token1: createLpObj.token1,
                fee: createLpObj.fee,
                tickLower: createLpObj.tickLower,
                tickUpper: createLpObj.tickUpper,
                amount0Desired: createLpObj.token0Amount,
                amount1Desired: createLpObj.token1Amount,
                amount0Min: 0,
                amount1Min: 0,
                recipient: address(this),
                deadline: block.timestamp + (15 minutes)
            });
        (tokenId, , amount0, amount1) = nonfungiblePositionManager.mint(params);
        positionMap.store[tokenId] = Deposit({
            customerId: createLpObj.customerId,
            token0: createLpObj.token0,
            token1: createLpObj.token1
        });
        positionMap.keys.push(tokenId);
        positionMap.keyExists[tokenId] = true;
        return (tokenId, amount0, amount1);
    }

        function increaseLiquidityCurrentRange(INonfungiblePositionManager
nonfungiblePositionManager, uint256 tokenId, uint256 amountAdd0, uint256 amountAdd1)
internal returns (uint256 amount0, uint256 amount1) {
        INonfungiblePositionManager.IncreaseLiquidityParams memory params =
                            INonfungiblePositionManager.IncreaseLiquidityParams({
                tokenId: tokenId,
                amount0Desired: amountAdd0,
                amount1Desired: amountAdd1,
                amount0Min: 0,
                amount1Min: 0,
                deadline: block.timestamp + (15 minutes)
            });
        (, amount0, amount1) = nonfungiblePositionManager.increaseLiquidity(params);
        return (amount0, amount1);
    }

        function removeAllPositionById(uint256 tokenId, INonfungiblePositionManager
nonfungiblePositionManager) internal returns (uint256 amount0, uint256 amount1) {
        return
nonfungiblePositionManager.decreaseLiquidity(INonfungiblePositionManager.DecreaseLiqui
dityParams({
            tokenId: tokenId,
            liquidity: queryLiquidityById(tokenId, nonfungiblePositionManager),
            amount0Min: 0,
            amount1Min: 0,
            deadline: block.timestamp + (15 minutes)
        }));
    }
```

Solution

It is recommended to set up slippage protection.

Status

Fixed

[N2] [Low] Withdrawal amount is inaccurate

Category: Design Logic Audit

Content

In the `mintPosition` function and `increaseLiquidity` function of the Vault contract, when the `token0Amount` and `token1Amount` parameters are not 0, the `withdrawAllFromAave` function will be called to withdraw the balance of all specified tokens from AAVE, causing unnecessary revenue losses.

- contracts/Vault.sol#L96-L105,L113-L122

```
    function mintPosition(LiquidityHelper.CreateLpObject memory createLpObject)
public dispatcherCheck allowListCheck(createLpObject.token0)
allowListCheck(createLpObject.token1) {
        if (createLpObject.token0Amount == 0 || createLpObject.token1Amount == 0) {
            withdrawFromAaveForTrading(createLpObject.token0,
createLpObject.token0Amount);
            withdrawFromAaveForTrading(createLpObject.token1,
createLpObject.token1Amount);
        } else {
            withdrawAllFromAave(createLpObject.token0);
            withdrawAllFromAave(createLpObject.token1);
        }
        positionMap.mintNewPosition(createLpObject,
uniInfo.nonfungiblePositionManager, approveInfo.liquidityApproveMap);
    }

        function increaseLiquidity(uint256 positionId, uint256 token0Amount, uint256
token1Amount) external dispatcherCheck {
        if (token0Amount == 0 || token1Amount == 0) {
            withdrawFromAaveForTrading(positionMap.store[positionId].token0,
token0Amount);
            withdrawFromAaveForTrading(positionMap.store[positionId].token1,
token1Amount);
        } else {
            withdrawAllFromAave(positionMap.store[positionId].token0);
            withdrawAllFromAave(positionMap.store[positionId].token1);
        }

LiquidityHelper.increaseLiquidityCurrentRange(uniInfo.nonfungiblePositionManager,
positionId, token0Amount, token1Amount);
    }
```

Solution

It is recommended to withdraw only the amount required.

Status

Fixed

[N3] [High] WETH tokens are not unwrapped

Category: Design Logic Audit

Content

In the `swapInputETHForToken` function of the Vault contract, the amount required to withdraw WETH from AAVE is used through the `withdrawFromAaveForTrading` function, but WETH is not unwrapped to ETH, and what is needed in the `swapInputETHForToken` function is ETH.

- contracts/Vault.sol#L75-L79

```
    function swapInputETHForToken(address tokenOut, uint24 fee, uint256 amountIn,
uint256 amountOutMin) external dispatcherCheck allowListCheck(tokenOut) returns
(uint256 amountOut) {
        withdrawFromAaveForTrading(uniInfo.WETH, amountIn);
        amountOut = SwapHelper.swapInputETHForToken(tokenOut, fee, amountIn,
amountOutMin, uniInfo.swapRouter, uniInfo.WETH);
        return tradingInfo.collectTradingFee(amountOut,
tradingInfo.swapTradingFeeRate, tokenOut);
    }
```

Solution

It is recommended to add corresponding logic to the function to unwrapped WETH to ETH.

Status

Fixed

[N4] [Suggestion] Missing event record

Category: Others

Content

Missing events for state changes in the contract.

- contracts/Vault.sol

```
setDispatcher
setSwapAllowList
updateTradingFee
setAutoStake
```

Solution

It is recommended to record events.

Status

Acknowledged

[N5] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the Vault contract, the Owner role and Dispatcher role can modify important parameters in the contract, and the Owner role can also withdraw all assets in the contract.The Dispatcher role can withdraw any amount of WETH through the `depositGasToDispatcher` function.

- contracts/Vault.sol

```
updateVaultName
setDispatcher
setSwapAllowList
updateTradingFee
setAutoStake
claimRewards
withdrawErc721NFT
withdrawTokens
withdrawETH
depositGasToDispatcher
```

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. And 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 side stated that in the protocol, every user creates their own Vault using the VaultFactory contract, which grants them full control over their assets. The Owner role is the user's own address.The project team has improved the permissions of the depositGasToDispatcher function.

[N6] [Suggestion] Redundant code

Category: Others

Content

In the Vault contract,ReentrancyGuard interface is not used.

- contracts/Vault.sol#L14

```
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

Solution

It is recommended to clarify the business involved. If this part of the logic is not needed, the code can be deleted to save gas, and at the same time, ensure that the set parameters are useful.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002311290001	SlowMist Security Team	2023.11.27 - 2023.11.29	High 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 2 high risk, 1 medium risk, 1 low risk, 2 suggestion vulnerabilities.

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