Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2021.10.08, the SlowMist security team received the ZENLINK team's security audit application for ZENLINK, 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:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit Version:

https://github.com/zenlinkpro/zenlink-evm-contracts

commit:bdf42e29c0aed1fec1219a63d4bdf5b073764419

Fixed Version:

https://github.com/zenlinkpro/zenlink-evm-contracts

commit:6194da2a10781f5f6fb74e011f3522442a2b44e0

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability	Low	Fixed
N2	Risk of excessive authority	Authority Control Vulnerability	Low	Fixed
N3	GasToken attack	Others	Suggestion	Confirmed
N4	Gas Optimization	Gas Optimization Audit	Suggestion	Fixed
N5	Token compatibility security reminder	Others	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

Factory:

https://moonriver.moonscan.io/address/0xf36AE63d89983E3aeA8AaaD1086C3280eb01438D#code

Router:

https://moonriver.moonscan.io/address/0xe6FE3Db4c5A2e4a9Ab3301201b38724E578B35cA#code

Stake:

https://moonriver.moonscan.io/address/0xF8Ea8152914df71a09eA29Be1462aC033d31E493#code

Bootstrap:

https://moonriver.moonscan.io/address/0x014E061549f72Ea6810F00aFb380CF0928B86b3c#code

Pair:

https://moonriver.moonscan.io/address/0x042e54b2b28265a7ce171f97391334bd47fe384c#code

4.2 Visibility Description

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

Factory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
createPair	External	Can Modify State	-
setBootstrap	External	Can Modify State	onlyAdmin
lockPairCreate	External	Can Modify State	onlyAdmin
unlockPairCreate	External	Can Modify State	onlyAdmin
setFeeto	External	Can Modify State	onlyAdmin
setFeeBasePoint	External	Can Modify State	onlyAdmin
lockPairMint	External	Can Modify State	onlyAdmin
unlockPairMint	External	Can Modify State	onlyAdmin
lockPairBurn	External	Can Modify State	onlyAdmin
unlockPairBurn	External	Can Modify State	onlyAdmin
lockPairSwap	External	Can Modify State	onlyAdmin
unlockPairSwap	External	Can Modify State	onlyAdmin

Pair			
Function Name	Visibility	Mutability	Modifiers

Pair			
lockMint	External	Can Modify State	-
unlockMint	External	Can Modify State	-
lockBurn	External	Can Modify State	-
unlockBurn	External	Can Modify State	-
lockSwap	External	Can Modify State	-
unlockSwap	External	Can Modify State	-
_safeTransfer	Private	Can Modify State	-
getReserves	Public	-	-
<Constructor>	Public	Can Modify State	ERC20
initialize	External	Can Modify State	-
_mintFee	Private	Can Modify State	-
mint	External	Can Modify State	lock mintUnlock
burn	External	Can Modify State	lock burnUnlock
swap	External	Can Modify State	lock swapUnlock
_update	Private	Can Modify State	-

AdminUpgradeable			
Function Name	Visibility	Mutability	Modifiers
_initializeAdmin	Internal	Can Modify State	-
candidateConfirm	External	Can Modify State	-

AdminUpgradeable			
setAdminCandidate	External	Can Modify State	onlyAdmin

Bootstrap			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setMinumAmount0	External	Can Modify State	whenNotEnded onlyAdmin
setMinumAmount1	External	Can Modify State	whenNotEnded onlyAdmin
setEndBlock	External	Can Modify State	whenNotEnded onlyAdmin
getUserInfo	External	-	-
getTotalLiquidity	Public	-	-
getExactLiquidity	Public	-	-
getLiquidityBalance	External	-	-
addProvision	External	Can Modify State	whenNotEnded nonReentrant
mintLiquidity	External	Can Modify State	whenEndedAndCapped nonReentrant onlyAdmin
claim	External	Can Modify State	whenEndedAndCapped whenLiquidityMinted nonReentrant
refund	External	Can Modify State	whenEndedAndFailed nonReentrant
withdraw	External	Can Modify State	onlyAdmin

Router			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
addLiquidity	Public	Can Modify State	ensure
addLiquiditySingleToken	External	Can Modify State	ensure
addLiquidityNativeCurrency	External	Payable	ensure
_addLiquidity	Private	Can Modify State	-
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityNativeCurrency	Public	Can Modify State	ensure
_swap	Private	Can Modify State	-
swapExactTokensForTokens	Public	Can Modify State	ensure
swapTokensForExactTokens	Public	Can Modify State	ensure
swapExactNativeCurrencyForTokens	External	Payable	ensure
swapTokensForExactNativeCurrency	External	Can Modify State	ensure
swapExactTokensForNativeCurrency	External	Can Modify State	ensure
swapNativeCurrencyForExactTokens	External	Payable	ensure
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

Stake			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
addReward	External	Can Modify State	onlyAdmin beforeEndPeriod
removeReward	External	Can Modify State	onlyAdmin beforeEndPeriod
withdraw	External	Can Modify State	onlyAdmin
setBlackList	External	Can Modify State	onlyAdmin
removeBlackList	External	Can Modify State	onlyAdmin
getStakerInfo	External	-	-
pauseStake	External	Can Modify State	onlyAdmin
unpauseStake	External	Can Modify State	onlyAdmin
pauseRedeem	External	Can Modify State	onlyAdmin
unpauseRedeem	External	Can Modify State	onlyAdmin
pauseClaim	External	Can Modify State	onlyAdmin
unpauseClaim	External	Can Modify State	onlyAdmin
stake	External	Can Modify State	beforeEndPeriod nonReentrant whenStakeNotPaused
redeem	External	Can Modify State	nonReentrant whenRedeemNotPaused

Stake			
getEstimatedRewardsBalance	External	-	-
claim	External	Can Modify State	nonReentrant whenClaimNotPaused

4.3 Vulnerability Summary

[N1] [Low] Risk of excessive authority

Category: Authority Control Vulnerability

Content

Admin can modify the configuration of the contract, including the function of locking and unlocking the contract, as well as adding and modifying the address of Bootstrap, and there is no event record for modifying the contract parameters, which is not conducive to the review of community users, and there is a risk of excessive authority.

code location:contracts/core/Factory.sol #L57-L108

```
function setBootstrap(address tokenA, address tokenB, address bootstrap) external
onlyAdmin {
        require(getPair[tokenA][tokenB] == address(0), "Factory: PAIR_EXISTS");
        getBootstrap[tokenA][tokenB] = bootstrap;
        getBootstrap[tokenB][tokenA] = bootstrap;
    }

    function lockPairCreate() external onlyAdmin {
        lockForPairCreate = true;
    }

    function unlockPairCreate() external onlyAdmin {
        lockForPairCreate = false;
    }

    function setFeeto(address _feeto) external onlyAdmin {
        feeto = _feeto;
    }
```

```
    function setFeeBasePoint(uint8 _basePoint) external onlyAdmin {
        require(_basePoint <= 30, "FORBIDDEN");
        feeBasePoint = _basePoint;
    }

    function lockPairMint(address tokenA, address tokenB) external onlyAdmin {
        address pair = getPair[tokenA][tokenB];
        IPair(pair).lockMint();
    }

    function unlockPairMint(address tokenA, address tokenB) external onlyAdmin {
        address pair = getPair[tokenA][tokenB];
        IPair(pair).unlockMint();
    }

    function lockPairBurn(address tokenA, address tokenB) external onlyAdmin {
        address pair = getPair[tokenA][tokenB];
        IPair(pair).lockBurn();
    }

    function unlockPairBurn(address tokenA, address tokenB) external onlyAdmin {
        address pair = getPair[tokenA][tokenB];
        IPair(pair).unlockBurn();
    }

    function lockPairSwap(address tokenA, address tokenB) external onlyAdmin {
        address pair = getPair[tokenA][tokenB];
        IPair(pair).lockSwap();
    }

    function unlockPairSwap(address tokenA, address tokenB) external onlyAdmin {
        address pair = getPair[tokenA][tokenB];
        IPair(pair).unlockSwap();
    }
```

code location:contracts/periphery/Bootstrap.sol #L101-L124

```
function setMinumAmount0(uint256 amount0)
        external
        whenNotEnded
        onlyAdmin
    {
```

```
        MINUM_AMOUNT0 = amount0;
    }

    function setMinumAmount1(uint256 amount1)
        external
        whenNotEnded
        onlyAdmin
    {
        MINUM_AMOUNT1 = amount1;
    }

    function setEndBlock(uint256 endBlock)
        external
        whenNotEnded
        onlyAdmin
    {
        require(endBlock > block.number, 'INVALID_END_BLOCK');
        END_BLOCK = endBlock;
    }
```

code location:contracts/periphery/Stake.sol #L146-L174

```
    function pauseStake() external onlyAdmin {
        require(!_stakePaused, 'STAKE_PAUSED');
        _stakePaused = true;
    }

    function unpauseStake() external onlyAdmin {
        require(_stakePaused, 'STAKE_UNPAUSED');
        _stakePaused = false;
    }

    function pauseRedeem() external onlyAdmin {
        require(!_redeemPaused, 'REDEEM_PAUSED');
        _redeemPaused = true;
    }

    function unpauseRedeem() external onlyAdmin {
        require(_redeemPaused, 'REDEEM_UNPAUSED');
        _redeemPaused = false;
    }
```

```
function pauseClaim() external onlyAdmin {
    require(!_claimPaused, 'CLAIM_PAUSED');
    _claimPaused = true;
}

function unpauseClaim() external onlyAdmin {
    require(_claimPaused, 'CLAIM_UNPAUSED');
    _claimPaused = false;
}
```

The Admin can set a blacklist, which is set as the address of the blacklist. Stake, redeem, and claim are not allowed. However, if ordinary users become blacklisted after the stake, they cannot withdraw their assets, so the Admin has the risk of excessive authority.

code location:contracts/periphery/Stake.sol #L128-L134

```
function setBlackList(address blacklistAddress) external onlyAdmin {
    _stakerInfos[blacklistAddress].inBlackList = true;
}

function removeBlackList(address blacklistAddress) external onlyAdmin {
    _stakerInfos[blacklistAddress].inBlackList = false;
}
```

The Admin can mint arbitrarily, and there is no upper limit.

code location:contracts/tokens/ZenlinkToken.sol #L57-L59

```
function mint(uint256 mintAmount) external onlyAdmin {
    _mint(msg.sender, mintAmount);
}
```

Solution

It is recommended to transfer the authority of the admin role to the timelock or governance contract, and add events to the function that modifies the contract parameters for recording. The _feeto address recommends using a multi-sign contract to avoid the leakage of the private key and the theft of the team's revenue.

Status

Fixed; 1.The project team deleted some functions with excessive permissions.

2.Event records for modifying contract parameters have been added.

[N2] [Low] Risk of excessive authority

Category: Authority Control Vulnerability

Content

Admin can extract the user's assets in the contract, and does not update totalAmount0, totalAmount1, which will cause the balance of the contract to be different from the accounting data.

code location:contracts/periphery/Bootstrap.sol #L274-L290

```
    function withdraw(
        address token,
        address to,
        uint256 amount
    ) external onlyAdmin {
        if (token == token0) {
            uint256 token0Balance = IERC20(token0).balanceOf(address(this));
            require(token0Balance.sub(amount) >= totalAmount0,
'INSUFFICIENT_TOKEN_BALANCE');
        }
        if (token == token1) {
            uint256 token1Balance = IERC20(token1).balanceOf(address(this));
            require(token1Balance.sub(amount) >= totalAmount1,
'INSUFFICIENT_TOKEN_BALANCE');
        }
        Helper.safeTransfer(token, to, amount);

        emit Withdraw(token, to, amount);
    }
```

After Admin raised REWARD_TOKEN, the totalRewardAmount is not updated, which will cause the data of totalRewardAmount to be inconsistent with the actual balance.

code location:contracts/periphery/Stake.sol #L118-L126

```
    function withdraw(address token, address to, uint256 amount) external onlyAdmin {
        if (token == REWARD_TOKEN) {
            uint256 rewardBalance = IERC20(REWARD_TOKEN).balanceOf(address(this));
            require(rewardBalance.sub(amount) >= totalRewardAmount,
'INSUFFICIENT_REWARD_BALANCE');
        }
        Helper.safeTransfer(token, to, amount);

        emit Withdraw(token, to, amount);
    }
```

Solution

After communication, the project party explained that the original intention of this design is that if the user mistakenly transfers tokens directly to our contract, then we can take the tokens out of the contract and return it to the user.This operation is to prevent losses by removing the reward tokens quickly if there is a hacking incident.

Status

Fixed; The project team changed the function name.

[N3] [Suggestion] GasToken attack

Category: Others

Content

The safeTransferNativeCurrency function does not limit the gaslimit of the call. If the to address is a third-party address entered by the user, there will be a gas token attack.

code location:contracts/libraries/Helper.sol #L87-L93

```
    function safeTransferNativeCurrency(address to, uint256 value) internal {
        (bool success, ) = to.call{value: value}(new bytes(0));
        require(
            success,
            "TransferHelper::safeTransferNativeCurrency: NativeCurrency transfer
failed"
        );
    }
```

code location:contracts/periphery/Router.sol #L209-L234

```
    function removeLiquidityNativeCurrency(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountNativeCurrencyMin,
        address to,
        uint256 deadline
    )
        public
        override
        ensure(deadline)
        returns (uint256 amountToken, uint256 amountNativeCurrency)
    {
        (amountToken, amountNativeCurrency) = removeLiquidity(
            token,
            WNativeCurrency,
            liquidity,
            amountTokenMin,
            amountNativeCurrencyMin,
            address(this),
            deadline
        );
        Helper.safeTransfer(token, to, amountToken);
        IWNativeCurrency(WNativeCurrency).withdraw(amountNativeCurrency);
        Helper.safeTransferNativeCurrency(to, amountNativeCurrency);
    }
```

code location:contracts/periphery/Router.sol #L326-L375

```
    function swapTokensForExactNativeCurrency(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external override ensure(deadline) returns (uint256[] memory amounts) {
        require(
            path[path.length - 1] == WNativeCurrency,
            "Router: INVALID_PATH"
```

```
        );
        amounts = Helper.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, "Router: EXCESSIVE_INPUT_AMOUNT");
        Helper.safeTransferFrom(
            path[0],
            msg.sender,
            Helper.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, address(this));
        IWNativeCurrency(WNativeCurrency).withdraw(amounts[amounts.length - 1]);
        Helper.safeTransferNativeCurrency(to, amounts[amounts.length - 1]);
    }

    function swapExactTokensForNativeCurrency(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external override ensure(deadline) returns (uint256[] memory amounts) {
        require(
            path[path.length - 1] == WNativeCurrency,
            "Router: INVALID_PATH"
        );
        amounts = Helper.getAmountsOut(factory, amountIn, path);
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "Router: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        Helper.safeTransferFrom(
            path[0],
            msg.sender,
            Helper.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, address(this));
        IWNativeCurrency(WNativeCurrency).withdraw(amounts[amounts.length - 1]);
        Helper.safeTransferNativeCurrency(to, amounts[amounts.length - 1]);
    }
```

Solution

It is recommended to limit the call gaslimit.

Reference:https://floriantramer.com/docs/slides/CESC18gastoken.pdf

Status

Confirmed

[N4] [Suggestion] Gas Optimization

Category: Gas Optimization Audit

Content

It is recommended to change assert to require to optimize gas, so as to avoid using up the remaining gas in the transaction after assert.

code location:contracts/periphery/Router.sol #L98-L182

```
    function addLiquidityNativeCurrency(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountNativeCurrencyMin,
        address to,
        uint256 deadline
    )
        external
        payable
        override
        ensure(deadline)
        returns (
            uint256 amountToken,
            uint256 amountNativeCurrency,
            uint256 liquidity
        )
    {
        (amountToken, amountNativeCurrency) = _addLiquidity(
            token,
            WNativeCurrency,
            amountTokenDesired,
            msg.value,
```

```
        amountTokenMin,
        amountNativeCurrencyMin
    );
    address pair = Helper.pairFor(factory, token, WNativeCurrency);
    Helper.safeTransferFrom(token, msg.sender, pair, amountToken);
    IWNativeCurrency(WNativeCurrency).deposit{
        value: amountNativeCurrency
    }();
    assert(IERC20(WNativeCurrency).transfer(pair, amountNativeCurrency));
    liquidity = IPair(pair).mint(to);
    if (msg.value > amountNativeCurrency)
        Helper.safeTransferNativeCurrency(
            msg.sender,
            msg.value - amountNativeCurrency
        ); // refund dust native currency, if any
}

function _addLiquidity(
    address token0,
    address token1,
    uint256 amount0Desired,
    uint256 amount1Desired,
    uint256 amount0Min,
    uint256 amount1Min
) private returns (uint256 amount0, uint256 amount1) {
    if (IFactory(factory).getPair(token0, token1) == address(0)) {
        IFactory(factory).createPair(token0, token1);
    }
    (uint256 reserve0, uint256 reserve1) = Helper.getReserves(
        factory,
        token0,
        token1
    );
    if (reserve0 == 0 && reserve1 == 0) {
        (amount0, amount1) = (amount0Desired, amount1Desired);
    } else {
        uint256 amount1Optimal = Helper.quote(
            amount0Desired,
            reserve0,
            reserve1
        );
        if (amount1Optimal <= amount1Desired) {
            require(
                amount1Optimal >= amount1Min,
```

```
                    "Router: INSUFFICIENT_1_AMOUNT"
                );
                (amount0, amount1) = (amount0Desired, amount1Optimal);
            } else {
                uint256 amount0Optimal = Helper.quote(
                    amount1Desired,
                    reserve1,
                    reserve0
                );
                assert(amount0Optimal <= amount0Desired);
                require(
                    amount0Optimal >= amount0Min,
                    "Router: INSUFFICIENT_0_AMOUNT"
                );
                (amount0, amount1) = (amount0Optimal, amount1Desired);
            }
        }
    }
```

code location:contracts/periphery/Router.sol #L27-L29

```
    receive() external payable {
        assert(msg.sender == WNativeCurrency); // only accept Native Currency via
fallback from the WNativeCurrency contract
    }
```

code location:contracts/periphery/Router.sol #L298-L324

```
    function swapExactNativeCurrencyForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        payable
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        require(path[0] == WNativeCurrency, "Router: INVALID_PATH");
```

```
        amounts = Helper.getAmountsOut(factory, msg.value, path);
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "Router: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        IWNativeCurrency(WNativeCurrency).deposit{value: amounts[0]}();
        assert(
            IERC20(WNativeCurrency).transfer(
                Helper.pairFor(factory, path[0], path[1]),
                amounts[0]
            )
        );
        _swap(amounts, path, to);
    }
```

code location:contracts/periphery/Router.sol #L377-L405

```
    function swapNativeCurrencyForExactTokens(
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        payable
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        require(path[0] == WNativeCurrency, "Router: INVALID_PATH");
        amounts = Helper.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, "Router: EXCESSIVE_INPUT_AMOUNT");
        IWNativeCurrency(WNativeCurrency).deposit{value: amounts[0]}();
        assert(
            IERC20(WNativeCurrency).transfer(
                Helper.pairFor(factory, path[0], path[1]),
                amounts[0]
            )
        );
        _swap(amounts, path, to);
        if (msg.value > amounts[0])
            Helper.safeTransferNativeCurrency(
```

```
            msg.sender,
            msg.value - amounts[0]
        ); // refund dust eth, if any
    }
```

Solution

It is recommended to change assert to require to optimize gas.

Status

Fixed; The project team has changed all assert to require.

[N5] [Suggestion] Token compatibility security reminder

Category: Others

Content

At present, the project does not check the actual balance of the account when transferring funds, but performs accounting according to the amount entered in the transfer, so it is not compatible with deflationary and inflationary tokens. The actual amount of deflationary and inflationary tokens in the transfer may be inconsistent with the incoming amount, which will cause accounting errors.

Solution

It is recommended to review the compatibility of the token with this project when accessing the token to ensure that there will be no compatibility issues.

Status

Confirmed; The project team has adopted the proposal.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002110180001	SlowMist Security Team	2021.10.08 - 2021.10.18	Low Risk

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 medium risk, 1 low risk, 3 suggestion vulnerabilities. And 1 medium risk, 2 suggestion vulnerabilities were confirmed and being fixed; All other findings were fixed. Regarding N1, although the project team deleted some functions with excessive permissions, they still retained some functions with excessive permissions, so the audit result was judged to be low risk. The code was 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