Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.06.13, the SlowMist security team received the MYX team's security audit application for MYX Protocol Phase4, 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

MYX Finance is a P2Pool2P decentralized perpetual contract protocol. MYX offers USDC-margined perpetual future contracts up to 50x leverage. This audit is based on the iterations since the previous audit.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Potential arbitrary address spoofing attack	Design Logic Audit	Critical	Fixed

NO	Title	Category	Level	Status
N2	Missing scope limit	Others	Low	Fixed
N3	call() should be used instead of transfer() and send()	Others	Suggestion	Fixed
N4	Missing return value check	Others	Suggestion	Fixed
N5	Risky calculation method for referralUserAmount	Design Logic Audit	Medium	Fixed
N6	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/d11-myx/myx-contracts

commit: 6681adb7ad637949438985b26464269a7efce5e8

Audit scope:

- The content of this audit covers the iterative changes since the previous audit commit f3ee5a369bb7fb294e97082cae7de8b94d465cd0.

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:

Router			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC2771Context
_msgSender	Internal	-	-
_msgData	Internal	-	-
_contextSuffixLength	Internal	-	-
salvageToken	External	Can Modify State	onlyPoolAdmin
setPaused	External	Can Modify State	onlyPoolAdmin
setUnPaused	External	Can Modify State	onlyPoolAdmin
updateIncreasePositionStatus	External	Can Modify State	onlyPoolAdmin
updateDecreasePositionStatus	External	Can Modify State	onlyPoolAdmin
updateOrderStatus	External	Can Modify State	onlyPoolAdmin
updateAddLiquidityStatus	External	Can Modify State	onlyPoolAdmin
updateRemoveLiquidityStatus	External	Can Modify State	onlyPoolAdmin
setEnterBacktracker	External	Can Modify State	onlyPoolAdmin
wrapWETH	External	Payable	-
getOperationStatus	External	-	-
setPriceAndAdjustCollateral	External	Payable	whenNotPaused nonReentrant validOperation
setPriceAndAdjustCollateralDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant validOperation
_setPriceAndAdjustCollateral	Private	Can Modify State	-

Router			
setPriceAndUpdateFundingRate	External	Payable	whenNotPaused nonReentrant
createIncreaseOrderWithTpSl	External	Payable	whenNotPaused nonReentrant
createIncreaseOrderWithTpSlDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_createIncreaseOrderWithTpSl	Private	Can Modify State	-
_createTpSl	Internal	Can Modify State	-
createIncreaseOrder	External	Payable	whenNotPaused nonReentrant
createIncreaseOrderDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_createIncreaseOrder	Private	Can Modify State	-
createDecreaseOrder	External	Payable	whenNotPaused nonReentrant
createDecreaseOrderDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_createDecreaseOrder	Private	Can Modify State	-
createDecreaseOrders	External	Payable	whenNotPaused nonReentrant
createDecreaseOrdersDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_createDecreaseOrders	Private	Can Modify State	-
_checkOrderAccount	Private	-	-
cancelOrder	External	Can Modify State	whenNotPaused nonReentrant validOperation
cancelOrderDelegate	External	Can Modify State	onlyForwarder whenNotPaused nonReentrant validOperation
_cancelOrder	Private	Can Modify State	-
cancelOrders	External	Can Modify State	whenNotPaused nonReentrant validOperation

Router			
cancelOrdersDelegate	External	Can Modify State	onlyForwarder whenNotPaused nonReentrant validOperation
_cancelOrders	Private	Can Modify State	-
addOrderTpSl	External	Payable	whenNotPaused nonReentrant
addOrderTpSlDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_addOrderTpSl	Private	Can Modify State	-
createTpSl	External	Payable	whenNotPaused nonReentrant
createTpSlDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_createTpSl	Private	Can Modify State	-
addLiquidityETH	External	Payable	whenNotPaused nonReentrant
addLiquidityETHDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_addLiquidityETH	Private	Can Modify State	-
addLiquidity	External	Payable	whenNotPaused nonReentrant
addLiquidityDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_addLiquidity	Private	Can Modify State	-
addLiquidityForAccount	External	Payable	whenNotPaused nonReentrant
addLiquidityForAccountDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_addLiquidityForAccount	Private	Can Modify State	-
removeLiquidity	External	Payable	whenNotPaused nonReentrant
removeLiquidityDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant

Router			
_removeLiquidity	Private	Can Modify State	-
removeLiquidityForAccount	External	Payable	whenNotPaused nonReentrant
removeLiquidityForAccountDelegate	External	Payable	onlyForwarder whenNotPaused nonReentrant
_removeLiquidityForAccount	Private	Can Modify State	-
claimReferralFeeDelegate	External	Can Modify State	onlyForwarder whenNotPaused nonReentrant
claimUserTradingFeeDelegate	External	Can Modify State	onlyForwarder whenNotPaused nonReentrant
_transferGasFee	Private	Can Modify State	-
removeLiquidityCallback	External	Can Modify State	onlyPool
createOrderCallback	External	Can Modify State	onlyOrderManager
addLiquidityCallback	External	Can Modify State	onlyPool

TrustedForwarder			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC2771Forwarder Roleable
<Receive Ether>	External	Payable	-
updateRouter	External	Can Modify State	onlyPoolAdmin
updateGasFee	External	Can Modify State	onlyPoolAdmin
updateGasToken	External	Can Modify State	onlyPoolAdmin
setUserRelayer	External	Can Modify State	-
setPublicRelayer	External	Can Modify State	onlyPoolAdmin
isUserRelayer	Public	-	-

TrustedForwarder			
execute	Public	Payable	-
executeBatch	Public	Payable	-
emergencyWithdrawETH	Public	Can Modify State	onlyAdmin
emergencyWithdrawERC20	Public	Can Modify State	onlyAdmin

GasStation			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Roleable
<Receive Ether>	External	Payable	-
swap	External	Payable	nonReentrant
emergencyWithdrawETH	Public	Can Modify State	onlyAdmin
emergencyWithdrawERC20	Public	Can Modify State	onlyAdmin

FeeCollector			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	reinitializer
getTradingFeeTier	External	-	-
getRegularTradingFeeTier	External	-	-
getKeeperNetworkFee	External	-	-
updatePositionManagerAddress	External	Can Modify State	onlyPoolAdmin
updateExecutionLogicAddress	External	Can Modify State	onlyPoolAdmin
updateRouterAddress	External	Can Modify State	onlyPoolAdmin
updateStakingPoolAddress	External	Can Modify State	onlyPoolAdmin

FeeCollector			
updatePoolAddress	External	Can Modify State	onlyPoolAdmin
updatePledgeAddress	External	Can Modify State	onlyPoolAdmin
updateTradingFeeTiers	External	Can Modify State	onlyPoolAdmin
updateTradingFeeTier	External	Can Modify State	onlyPoolAdmin
updateMaxReferralsRatio	External	Can Modify State	onlyPoolAdmin
claimStakingTradingFee	External	Can Modify State	onlyStakingPool
claimTreasuryFee	External	Can Modify State	onlyTreasury
claimReferralFee	External	Can Modify State	nonReentrant
claimReferralFeeDelegate	External	Can Modify State	nonReentrant onlyRouter
claimUserTradingFee	External	Can Modify State	nonReentrant
claimUserTradingFeeDelegate	External	Can Modify State	nonReentrant onlyRouter
claimKeeperTradingFee	External	Can Modify State	nonReentrant
claimKeeperNetworkFee	External	Can Modify State	nonReentrant
distributeTradingFee	External	Can Modify State	onlyPositionManagerOrLogic
distributeNetworkFee	External	Can Modify State	onlyPositionManagerOrLogic
_collectTradingFee	Internal	Can Modify State	-
_updateTradingFeeTier	Internal	Can Modify State	-
rescueKeeperNetworkFee	External	Can Modify State	nonReentrant onlyAdmin

PositionManager			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	reinitializer
setRouter	External	Can Modify State	onlyPoolAdmin

PositionManager			
increasePosition	External	Can Modify State	onlyExecutor
decreasePosition	External	Can Modify State	onlyExecutor
adjustCollateral	External	Can Modify State	onlyRouter
updateFundingRate	External	Can Modify State	onlyRouter
_takeFundingFeeAddTraderFee	Internal	Can Modify State	-
_settleLPPosition	Internal	Can Modify State	-
_calLpProfit	Internal	Can Modify State	-
_handleCollateral	Internal	Can Modify State	-
_regularTradingFee	Internal	-	-
_updateFundingRate	Internal	Can Modify State	-
_nextFundingRate	Internal	-	-
getTradingFee	Public	-	-
getFundingFee	Public	-	-
getCurrentFundingRate	External	-	-
getNextFundingRate	External	-	-
getNextFundingRateUpdateTime	External	-	-
needADL	External	-	-
needLiquidation	Public	-	-
exposureAmountChecker	External	-	-
maxAvailableLiquidity	Public	-	-
getExposedPositions	Public	-	-
getPosition	Public	-	-

PositionManager			
getPositionByKey	Public	-	-
getPositionKey	Public	-	-
_max	Internal	-	-
_min	Internal	-	-

Pool			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	reinitializer
<Receive Ether>	External	Payable	-
_unwrapWETH	Private	Can Modify State	-
setPoolView	External	Can Modify State	onlyPoolAdmin
setSpotSwap	External	Can Modify State	onlyPoolAdmin
setRiskReserve	External	Can Modify State	onlyPoolAdmin
setFeeCollector	External	Can Modify State	onlyPoolAdmin
setPositionManager	External	Can Modify State	onlyPoolAdmin
setOrderManager	External	Can Modify State	onlyPoolAdmin
setRouter	External	Can Modify State	onlyPoolAdmin
addStableToken	External	Can Modify State	onlyPoolAdmin
removeStableToken	External	Can Modify State	onlyPoolAdmin
addPair	External	Can Modify State	onlyPoolAdmin

Pool			
updatePair	External	Can Modify State	onlyPoolAdmin
updateTradingConfig	External	Can Modify State	onlyPoolAdmin
updateTradingFeeConfig	External	Can Modify State	onlyPoolAdmin
_increaseTotalAmount	Internal	Can Modify State	-
_decreaseTotalAmount	Internal	Can Modify State	-
increaseReserveAmount	External	Can Modify State	onlyPositionManager
decreaseReserveAmount	External	Can Modify State	onlyPositionManager
updateAveragePrice	External	Can Modify State	onlyPositionManager
setLPStableProfit	External	Can Modify State	onlyPositionManagerOrFeeCollector
givebackTradingFee	External	Can Modify State	onlyFeeCollector
getAvailableLiquidity	External	-	-
addLiquidity	External	Can Modify State	onlyRouter
removeLiquidity	External	Can Modify State	onlyRouter
_transferToken	Internal	Can Modify State	-
_swapInUni	Private	Can Modify State	-
_getStableTotalAmount	Internal	-	-
_getIndexTotalAmount	Internal	-	-
_addLiquidity	Private	Can Modify State	-
_removeLiquidity	Private	Can Modify State	-

Pool			
claimFee	External	Can Modify State	onlyTreasury
transferTokenTo	External	Can Modify State	transferAllowed
transferEthTo	External	Can Modify State	transferAllowed
transferTokenOrSwap	External	Can Modify State	transferAllowed
emergencyWithdrawETH	Public	Can Modify State	onlyAdmin
emergencyWithdrawERC20	Public	Can Modify State	onlyAdmin
getLpPnl	Public	-	-
lpProfit	Public	-	-
getVault	Public	-	-
getPair	Public	-	-
getTradingConfig	External	-	-
getTradingFeeConfig	External	-	-

4.3 Vulnerability Summary

[N1] [Critical] Potential arbitrary address spoofing attack

Category: Design Logic Audit

Content

In the Router contract,the delegated trading feature implements the ERC2771 standard, which permits users to interact with contracts indirectly via an intermediary forwarder contract. This mechanism allows designated accounts to cover gas fees and perform trades on behalf of the users, enhancing the overall user experience and accessibility of the system.

However, since the Router contract references the Multicall contract and the multicall function does not check non-canonical contexts (i.e., msg.sender is not _msgSender()), a malicious user could potentially construct invalid relay request data (request.data) and call the multicall function of the Router contract through the Forwarder contract. The multicall function could then be used to invoke other functions with delegated transaction capabilities. This would allow the attacker to forge _msgSender() as other legitimate users and consume their funds without their consent.

You can refer to the following analysis articles about arbitrary address spoofing attacks:

https://slowmist.medium.com/an-in-depth-analysis-of-arbitrary-address-spoofing-attacks-5e6df80d5dae

https://blog.thirdweb.com/vulnerability-report/

Code Location: contracts/libraries/Multicall.sol

```
    function multicall(bytes[] calldata data) external payable override returns
(bytes[] memory results) {
        results = new bytes[](data.length);
        for (uint256 i = 0; i < data.length; i++) {
            (bool success, bytes memory result) = address(this).delegatecall(data[i]);
            if (!success) {
                if (result.length < 68) revert();
                assembly {
                    result := add(result, 0x04)
                }
                revert(abi.decode(result, (string)));
            }

            results[i] = result;
        }
    }
}
```

Solution

It is recommended to use the multicall contract from the latest version of the OpenZeppelin library.

Status

Fixed; Fixed in commit 264dd0b904b1d9078e1ea2ff390e9a8777248df2.

[N2] [Low] Missing scope limit

Category: Others

Content

In the TrustedForwarder contract, the PoolAdmin role can set the gas fee that needs to be charged when delegating trades by calling the updateGasFee function . However, there is no limit to the range of the gasFee, so if the PoolAdmin role sets this value too high, it will unexpectedly consume the user's funds.

Code Location: contracts/core/TrustedForwarder.sol

```
    function updateGasFee(uint256 _gasFee) external onlyPoolAdmin {
        uint256 old = gasFee;
        gasFee = _gasFee;
        emit UpdateGasFee(msg.sender, old, _gasFee);
    }
```

Solution

It is recommended to add the range checking to the corresponding functions.

Status

Fixed; Fixed in commit 264dd0b904b1d9078e1ea2ff390e9a8777248df2.

[N3] [Suggestion] call() should be used instead of transfer() and send()

Category: Others

Content

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

References:

https://consensys.io/diligence/blog/2019/09/stop-using-soliditys-transfer-now/

Code Location: contracts/periphery/GasStation.sol

```
    function swap(
        address[] calldata tokens,
        bytes[] calldata updateData,
        uint64[] calldata publishTimes,
        address recipient,
        uint256 stableAmount
    ) external nonReentrant payable {
        ...

        payable(recipient).transfer(ethAmount);
        ...
    }

    function emergencyWithdrawETH(address recipient) public onlyAdmin {
        payable(recipient).transfer(address(this).balance);
    }
```

contracts/periphery/MultipleTransfer.sol

```
    function batchTransferETH(
        address[] memory recipients,
        uint256[] memory amounts,
        bytes32[][] memory proofs
    ) public onlyOperator whenNotPaused {
        require(recipients.length == amounts.length, "Recipients and amounts length
 mismatch");

        for (uint i = 0; i < recipients.length; i++) {
            ...

            payable(recipients[i]).transfer(amounts[i]);
        }
    }

    function emergencyWithdrawETH(address recipient) public onlyOwner {
        payable(recipient).transfer(address(this).balance);
    }
```

contracts/pool/Pool.sol

```
    function emergencyWithdrawETH(address recipient) public onlyAdmin {
        payable(recipient).transfer(address(this).balance);
    }
```

contracts/core/TrustedForwarder.sol

```
function emergencyWithdrawETH(address recipient) public onlyAdmin {
    payable(recipient).transfer(address(this).balance);
}
```

contracts/core/Router.sol

```
function _createDecreaseOrders(
    address sender,
    TradingTypes.DecreasePositionRequest[] memory requests
) private returns (uint256[] memory orderIds) {
    ...

    if (surplus > 0) {
        payable(sender).transfer(surplus);
    }
    return orderIds;
}
```

Solution

It is recommended to use call() instead of transfer(), but be sure to respect the CEI pattern or add re-entrancy guards, and the return value should be checked.

Status

Fixed; Fixed in commit 264dd0b904b1d9078e1ea2ff390e9a8777248df2.

[N4] [Suggestion] Missing return value check

Category: Others

Content

In the GasStation contract, the user can call the swap function to transfer tokens in the contract. But it does not check the return value. If external tokens do not adopt the EIP20 standard, it may lead to “false top-up” issues.

Code Location: contracts/periphery/GasStation.sol

```
function swap(
    address[] calldata tokens,
    bytes[] calldata updateData,
    uint64[] calldata publishTimes,
```

```
        address recipient,
        uint256 stableAmount
    ) external nonReentrant payable {
        require(interval[msg.sender] + 86400 <= block.timestamp, "next interval");

        stableToken.transferFrom(msg.sender, address(this), stableAmount);

        ...
    }
```

contracts/core/Router.sol

```
    function _transferGasFee(address sender) private {
        ...

        IERC20(forwarder.gasToken()).transferFrom(sender, address(forwarder), gasFee);
    }
```

Solution

It is recommended to add a check of the return value or use SafeERC20 library.

Status

Fixed; Fixed in commit 264dd0b904b1d9078e1ea2ff390e9a8777248df2.

[N5] [Medium] Risky calculation method for referralUserAmount

Category: Design Logic Audit

Content

In the FeeCollector contract, The distributeTradingFee function calculates the referral fee to be paid back to the user. However, the fee paid to the referralOwner role is calculated by subtracting referralUserAmount from referralsAmount. There is a case where, if referralsRatio > referralUserRatio > maxReferralsRatio, the value of referralsAmount may be smaller than the value of referralUserAmount. In this case, an error will occur when calculating the fee to be paid to the referralOwner role.

Code Location: contracts/core/FeeCollector.sol#L349-359

```
    function distributeTradingFee(
        IPool.Pair memory pair,
        address account,
        uint256 orderId,
```

```
        address keeper,
        uint256 size,
        uint256 sizeDelta,
        uint256 executionPrice,
        uint256 tradingFee,
        bool isMaker,
        TradingFeeTier memory tradingFeeTier,
        int256 exposureAmount,
        int256 afterExposureAmount,
        uint256 referralsRatio,
        uint256 referralUserRatio,
        address referralOwner
    ) external override onlyPositionManagerOrLogic returns (int256 lpAmount, int256
vipTradingFee, uint256 givebackFeeAmount) {
        ...

        if (referralOwner != address(0)) {
            referralsAmount = surplusFee.mulPercentage(
                Math.min(referralsRatio, maxReferralsRatio)
            );
            referralUserAmount = surplusFee.mulPercentage(Math.min(referralUserRatio,
referralsRatio));

            referralFee[account] += referralUserAmount;
            referralFee[referralOwner] += referralsAmount - referralUserAmount;

            surplusFee = surplusFee - referralsAmount;
        }

        ...
    }
```

Solution

It is recommended that when calculating the value of referralUserAmount, the minimum value between `referralUserRatio` and `referralsRatio` should be taken, and then the minimum value between this and `maxReferralsRatio` should be taken again. The final value should be used as the numerator in the calculation, and it must be ensured that the value of maxReferralsRatio cannot be greater than 0.5e8.

Status

Fixed; Fixed in commit 264dd0b904b1d9078e1ea2ff390e9a8777248df2.

[N6] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the TrustedForwarder contract, the onlyPoolAdmin role can modify the gasFee, the gasToken and set any address as the publicRelayer role. the Admin role can withdraw any tokens from the contract. If the private key of these roles is leaked, it will cause the loss of contract funds.

Code Location: contracts/core/TrustedForwarder.sol

```
    function updateGasFee(uint256 _gasFee) external onlyPoolAdmin {
        uint256 old = gasFee;
        gasFee = _gasFee;
        emit UpdateGasFee(msg.sender, old, _gasFee);
    }

    function updateGasToken(address _gasToken) external onlyPoolAdmin {
        address old = gasToken;
        gasToken = _gasToken;
        emit UpdateGasToken(msg.sender, old, _gasToken);
    }

    function setPublicRelayer(address relayer, bool enable) external onlyPoolAdmin {
        publicRelayer[relayer] = enable;
    }

    function emergencyWithdrawETH(address recipient) public onlyAdmin {
        payable(recipient).transfer(address(this).balance);
    }

    function emergencyWithdrawERC20(IERC20 token, address recipient) public onlyAdmin
{
        uint256 balance = token.balanceOf(address(this));
        token.safeTransfer(recipient, balance);
    }
```

2.In the GasStation contract, the Admin role can withdraw any tokens from the contract. If the private key of the role is leaked, it will cause the loss of contract funds.

Code Location: contracts/periphery/GasStation.sol

```
    function emergencyWithdrawETH(address recipient) public onlyAdmin {
        payable(recipient).transfer(address(this).balance);
    }
```

```
    function emergencyWithdrawERC20(IERC20 token, address recipient) public onlyAdmin
{
        uint256 balance = token.balanceOf(address(this));
        token.safeTransfer(recipient, balance);
    }
```

3.In the Pool contract, the Admin role can withdraw any tokens from the contract. If the private key of the role is leaked, it will cause the loss of contract funds.

Code Location: contracts/pool/Pool.sol

```
    function emergencyWithdrawETH(address recipient) public onlyAdmin {
        payable(recipient).transfer(address(this).balance);
    }

    function emergencyWithdrawERC20(IERC20 token, address recipient) public onlyAdmin
{
        uint256 balance = token.balanceOf(address(this));
        token.safeTransfer(recipient, balance);
    }
```

Solution

In the short term, transferring admin's 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 responded: They will redesign it in the next major version update.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002406180001	SlowMist Security Team	2024.06.13 - 2024.06.18	Medium 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 critical risk, 2 medium risk, 1 low risk, 2 suggestion vulnerabilities. All the findings were fixed and acknowledged. The code was not deployed to the mainnet. Since the code has not yet been deployed to the main network and the permissions of the core roles have not been transferred and managed, the reported risk level is medium risk for the time being.

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