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 2023.09.25, the SlowMist security team received the dappOS team's security audit application for DappOS Contracts Core, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
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

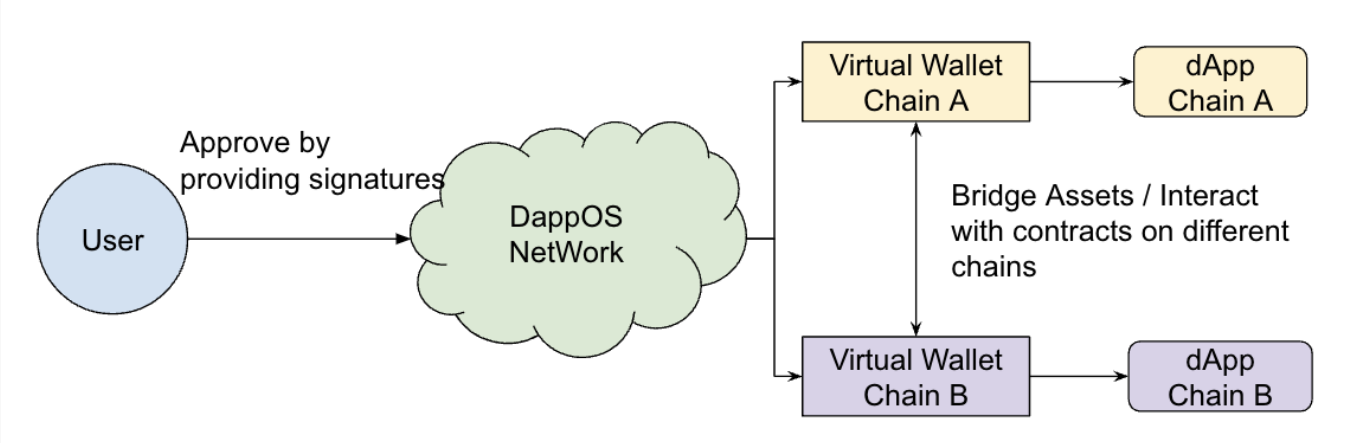
Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

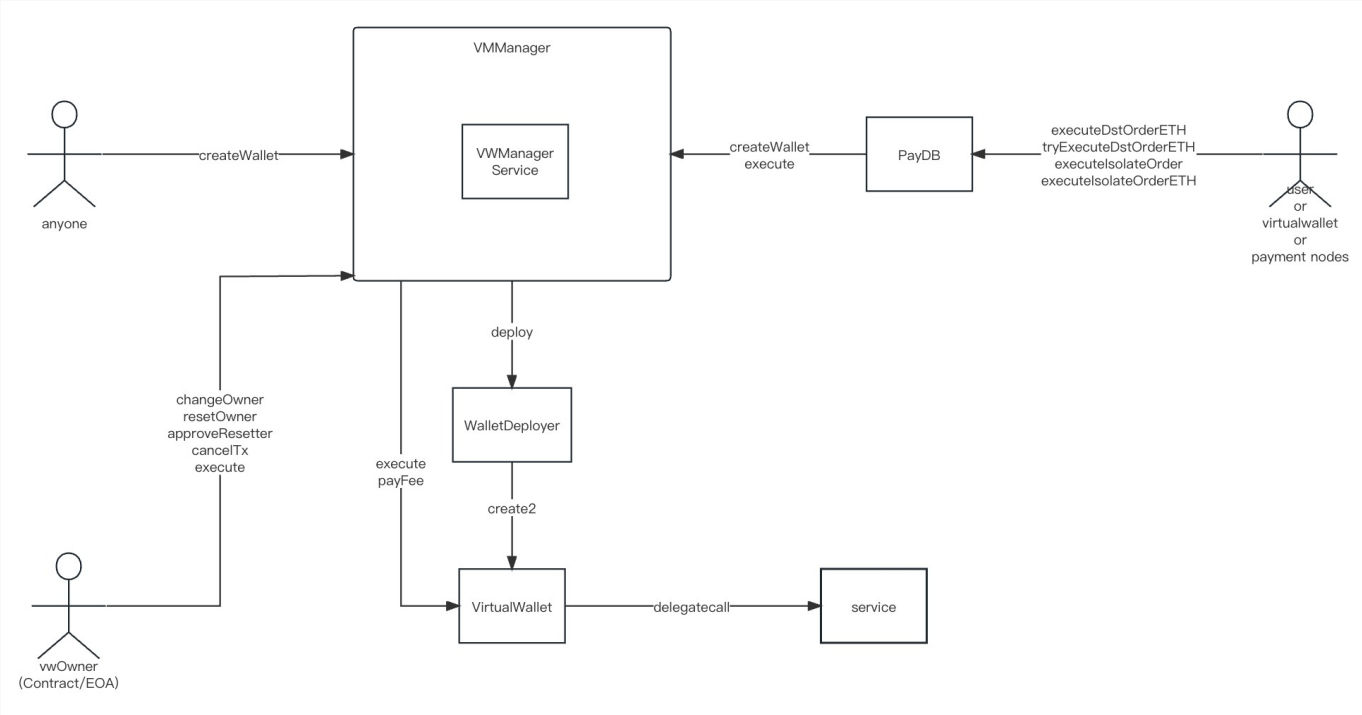
3.1 Project Introduction

An operating protocol designed to lower the barrier to interacting with crypto infrastructures.

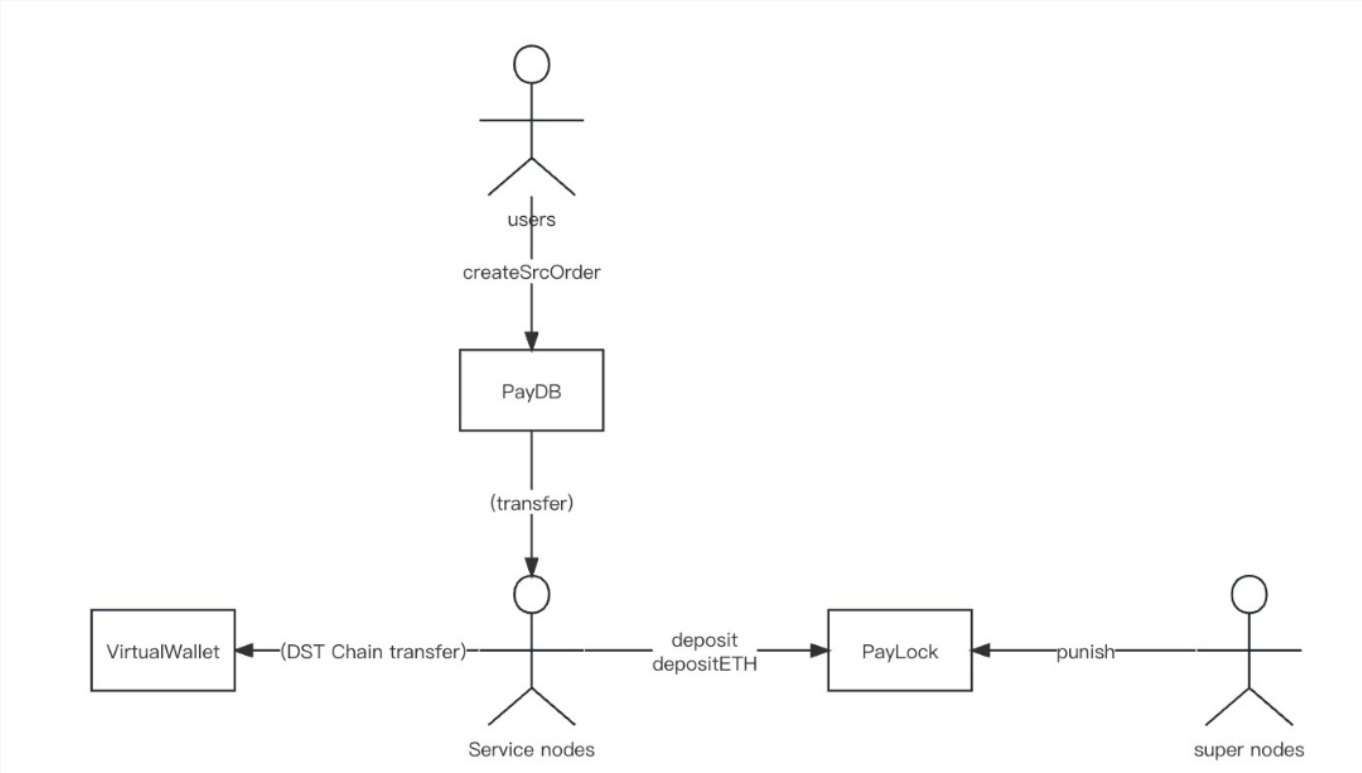
Core flowchart:



VMManager flowchart:



Assets flowchart:



3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Reentrancy risks	Reentrancy Vulnerability	Low	Acknowledged
N2	Unauthorized information status modification	Design Logic Audit	Low	Acknowledged
N3	Unverified feeReceiver parameter	Design Logic Audit	Low	Acknowledged
N4	Reentrancy risks	Reentrancy Vulnerability	Medium	Fixed
N5	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N6	Unverified manager and feeReceiver parameter	Design Logic Audit	Low	Acknowledged
N7	Redundant code	Others	Suggestion	Fixed
N8	Reentrancy risks	Reentrancy Vulnerability	Low	Fixed
N9	Insufficient WithdrawPendingTime error	Design Logic Audit	High	Fixed
N10	Unverified Node Mortgage Requirement	Design Logic Audit	High	Fixed
N11	Service nodes are at risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/DappOSDao/contracts-core

Initial audit commit: c929d0c0a6d61935a56639b3e220710ec9788283

Final audit commit: d2376c264bb240f325a79752e0b434149b7d2f7d

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:

VWManager			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	onlyOwner
_calDomain	Internal	-	-
verifyEIP1271Signature	External	-	-
verifyProof	Internal	Can Modify State	-
configSrcChain	External	Can Modify State	onlyOwner
requestConfigSrcChain	External	Can Modify State	onlyOwner
createWallet	External	Can Modify State	-
configFee	External	Can Modify State	onlyOwner
splitAndSendFee	Internal	Can Modify State	-
execute	External	Can Modify State	nonReentrant
storeInfo	External	Can Modify State	-
deleteInfo	External	Can Modify State	-
_getSignedDigest	Internal	-	-

VWManagerService			
Function Name	Visibility	Mutability	Modifiers
verifyOperation	Internal	-	-

VWManagerService			
cancelTx	External	Can Modify State	-
_walletPayFee	Internal	Can Modify State	-
changeOwner	External	Can Modify State	-
approveResetter	External	Can Modify State	-
resetOwner	External	Can Modify State	nonReentrant
_resetOwner	Internal	Can Modify State	-

VirtualWallet			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
execute	External	Can Modify State	onlyVWManager
payFee	External	Can Modify State	onlyVWManager
isValidSignature	External	-	-
onERC721Received	External	-	-
onERC1155Received	External	-	-
onERC1155BatchReceived	External	-	-
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-

WalletDeployer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deploy	External	Can Modify State	onlyVWManager

PayDB			
Function Name	Visibility	Mutability	Modifiers
createSrcOrder	External	Can Modify State	-
createSrcOrderETH	External	Payable	-
executeDstOrderETH	External	Payable	-
tryExecuteDstOrderETH	External	Payable	-
executeIsolateOrder	External	Can Modify State	-
executeIsolateOrderETH	External	Payable	-
cancelOrderETH	External	Payable	-
_executeIsolateOrder	Internal	Can Modify State	-
_createSrcOrder	Internal	Can Modify State	-
createWalletIfNotExists	Internal	Can Modify State	-
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-

PayLock			
Function Name	Visibility	Mutability	Modifiers
deposit	External	Can Modify State	-
depositETH	External	Payable	-
_deposit	Internal	Can Modify State	-
submitWithdrawRequest	External	Can Modify State	-
claim	External	Can Modify State	-
punish	External	Can Modify State	onlyOwner
configToken	External	Can Modify State	onlyOwner

PayLock			
configWithdrawPendingTime	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-
<Fallback>	External	Payable	-

4.3 Vulnerability Summary

[N1] [Low] Reentrancy risks

Category: Reentrancy Vulnerability

Content

In the PayDB contract, the `executeDstOrderETH`, `executeDstOrderETH`, `tryExecuteDstOrderETH`, `cancelOrderETH`, `_executeIsolateOrder`, and `_createSrcOrder` functions do not add anti-reentrancy locks, and there is a risk of reentrancy attacks when calling the safeTransferETH function.

- contracts/core/PayDB.sol#L75-L153,L162-L249,L287-L335,L337-L383,L385-L475

```
function executeDstOrderETH(
        address orderOwner,
        address receiver,
        address wallet,
        ExePayOrderParam[] calldata eparams,
        IVWManager.VWExecuteParam calldata vwExeParam
    ) external payable override{
    ````
     if (eparams[i].tokenOut == address(0)) {
                TransferHelper.safeTransferETH(
                    _realReceiver,
                    eparams[i].amountOut
                );
                totalETH += eparams[i].amountOut;
            } else {
                TransferHelper.safeTransferFrom(
                    eparams[i].tokenOut,
                    msg.sender,
                    _realReceiver,
                    eparams[i].amountOut
                );
            }
```
```

````
```
}

function tryExecuteDstOrderETH(
    address orderOwner,
    address receiver,
    address wallet,
    ExePayOrderParam[] calldata eparams,
    IVWManager.VWExecuteParam calldata vwExeParam
) external payable override {
```
if (eparams[i].tokenOut == address(0)) {
            TransferHelper.safeTransferETH(
                _realReceiver,
                eparams[i].amountOut
            );
            totalETH += eparams[i].amountOut;
        } else {
            TransferHelper.safeTransferFrom(
                eparams[i].tokenOut,
                msg.sender,
                _realReceiver,
                eparams[i].amountOut
            );
        }
```
}

 function cancelOrderETH(
    address sender,
    address receiver,
    CreatePayOrderParam[] calldata cparams,
    bytes32[] calldata workFlowHashs
) external payable override {
```
        require(msg.value == totalETH,"E18");
    if(totalETH > 0){
        TransferHelper.safeTransferETH(sender, msg.value);
    }
```
}

function _executeIsolateOrder(
    address receiver,
    address wallet,
    ExePayOrderParam[] calldata eparams,
    IVWManager.VWExecuteParam calldata vwExeParam
) internal {
```
````

```
 if (eparams[i].tokenOut == address(0)) {
            TransferHelper.safeTransferETH(
                receiver,
                eparams[i].amountOut
            );
            totalETH += eparams[i].amountOut;
        } else {
            TransferHelper.safeTransferFrom(
                eparams[i].tokenOut,
                msg.sender,
                receiver,
                eparams[i].amountOut
            );
        }
````
}

function _createSrcOrder(
    address _orderOwner,
    address wallet,
    address receiver,
    CreatePayOrderParam[] calldata cparams,
    VwOrderDetail calldata vwDetail,
    CallParam calldata callParam
) internal {
````
if (cparams[i].tokenIn == address(0)) {
            // Transfer ETH to node
            TransferHelper.safeTransferETH(
                cparams[i].node,
                cparams[i].amountIn
            );
            totalEth += cparams[i].amountIn;
        } else {
            // Transfer ERC20 to node
            TransferHelper.safeTransferFrom(
                cparams[i].tokenIn,
                msg.sender,
                cparams[i].node,
                cparams[i].amountIn
            );
        }
````
}
```

**Solution**

It's recommended to add the nonReentrant lock to prevent the unexpected callback reentrancy of the external calls.
```

Status

Acknowledged

[N2] [Low] Unauthorized information status modification

Category: Design Logic Audit

Content

In the `storeInfo` function of the VWManager contract, we found the following issues：

1. Any user can set the `willDelete` parameter of information stored by other users to false, thereby deleting `infoSender[infoHash]` so that it cannot be deleted.
2. Key Parameter Settings Unrecorded Events.
3. The function may be subject to MEV attacks, causing the user to store `infoSender[infoHash] = msg.sender` as the attacker's address when storing information.

- contracts/core/vwmanager/VWManager.sol#L295-L311

```
function storeInfo(
     bytes calldata info,
     bool willDelete
 ) external {
     if(info.length > 0){
         bytes32 infoHash = keccak256(info);
         if(eip1271Info[infoHash].length == 0){
             eip1271Info[infoHash] = info;
             emit InfoStored(infoHash, info);
             if(willDelete){
                 infoSender[infoHash] = msg.sender;
             }
         } else if(!willDelete) {
             delete infoSender[infoHash];
         }
     }
 }
```

Solution

It is recommended to re-evaluate whether the functions implemented by the function meet the design expectations.

Status

Acknowledged;

After communicating with the project team, it has been confirmed that the functionality of this function aligns with the intended design.

[N3] [Low] Unverified feeReceiver parameter

Category: Design Logic Audit

Content

The `cancelTx`, `changeOwner`, `approveResetter`, and `resetOwner` functions of the VWManagerService contract, the `feeReceiver` parameter is not verified. It may be subject to MEV attack risk. The attacker replaces the `feeReceiver` parameters, causing losses.

- contracts/core/vwmanager/VWManagerService.sol#L53-L92,L123-L161,L177-L217,L227-L256

```
    function cancelTx(
        uint256 code,
        address wallet,
        uint256 codeToCancel,
        FeeParam calldata fParam,
        bytes calldata signature
    ) external {
    ````
        bytes32 dataHash = keccak256(
            abi.encode(
                CANCEL_TYPEHASH,
                code,
                codeToCancel,
                fParam.gasToken,
                fParam.gasTokenPrice,
                fParam.priorityFee,
                fParam.gasLimit
)
        );

        result[walletOwner[wallet]][codeToCancel] = uint256(CodeStatus.CANCELED);
        result[walletOwner[wallet]][code] = uint256(CodeStatus.SUCCEED);
        emit TxCanceled(codeToCancel);

        verifyOperation(
            walletOwner[wallet],
            domainSeparator[srcChain],
            dataHash,
            signature,
            CANCEL_TYPEHASH
        );
```
```

```
        _walletPayFee(wallet, preGas, fParam);
    }

       function changeOwner(
        uint256 code,
        address wallet,
        address newOwner,
        FeeParam calldata fParam,
        bytes calldata signature
    ) external {
````
            bytes32 dataHash = keccak256(
            abi.encode(
                CHANGE_OWNER_TX_TYPEHASH,
                code,
                newOwner,
                fParam.gasToken,
                fParam.gasTokenPrice,
                fParam.priorityFee,
                fParam.gasLimit
)
        );
        address previousOwner = _resetOwner(wallet, newOwner);

        result[walletOwner[wallet]][code] = uint256(CodeStatus.SUCCEED);

        verifyOperation(
            previousOwner,
            domainSeparator[srcChain],
            dataHash,
            signature,
            CHANGE_OWNER_TX_TYPEHASH
        );
        _walletPayFee(wallet, preGas, fParam);
    }

     function approveResetter(
        uint256 code,
        address wallet,
        address resetter,
        bool approved,
        FeeParam calldata fParam,
        bytes calldata signature
) external returns (bytes32 dataHash) {
````
    dataHash = keccak256(
            abi.encode(
                RESETTER_APPROVE_TYPEHASH,
                code,
```

```
                resetter,
                approved,
                fParam.gasToken,
                fParam.gasTokenPrice,
                fParam.priorityFee,
                fParam.gasLimit
            )
        );
        approvedResetter[wallet] = approved ? resetter : address(0);
        emit ResetterChanged(approvedResetter[wallet]);
        result[walletOwner[wallet]][code] = uint256(CodeStatus.SUCCEED);

        verifyOperation(
            walletOwner[wallet],
            domainSeparator[srcChain],
            dataHash,
            signature,
            RESETTER_APPROVE_TYPEHASH
        );

        _walletPayFee(wallet, preGas, fParam);
    }

    function resetOwner(
        uint256 code,
        address wallet,
        address newOwner,
        FeeParam calldata fParam,
        bytes calldata data
    ) external nonReentrant{
````
    require (
            IVWResetter(approvedResetter[wallet]).verify(
                wallet,
                newOwner,
                data,
                fParam.gasToken,
                fParam.gasTokenPrice,
                fParam.priorityFee,
                fParam.gasLimit
            ), "E5");
        _resetOwner(wallet, newOwner);
        result[walletOwner[wallet]][code] = uint256(CodeStatus.SUCCEED);

        _walletPayFee(wallet, preGas, fParam);
    }
```
```

Solution

It is recommended to add verification of feeReceiver parameters.

Status

Acknowledged

[N4] [Medium] Reentrancy risks

Category: Reentrancy Vulnerability

Content

In the PayLock contract, the `deposit` function does not add an anti-reentrancy lock, and there is a risk of reentrancy attacks when calling the safeTransferFrom function.

- contracts/core/PayLock.sol#L73-L78

```
    function deposit(address token, uint amount, address node) external {
        uint256 beforeTransfer = IERC20(token).balanceOf(address(this));
        TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);
        uint256 afterTransfer = IERC20(token).balanceOf(address(this));
        _deposit(token,  afterTransfer - beforeTransfer, node);
    }
```

Solution

It's recommended to add the nonReentrant lock to prevent the unexpected callback reentrancy of the external calls.

Status

Fixed

[N5] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

Owner accounts can operate the key functions.

```
PayLock punish
PayLock configToken
PayLock configWithdrawPendingTime
VWManager       configFee
```

```
VWManager       requestConfigSrcChain
VWManager       configSrcChain
```

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

[N6] [Low] Unverified manager and feeReceiver parameter

Category: Design Logic Audit

Content

In the `verifyProof` function of the VWManager contract, only the 8 parameters in vweParam were verified, and the manager and `feeReceiver` parameters were not verified.

- contracts/core/vwmanager/VWManager.sol#L113-L145

```
 function verifyProof(uint resCode, address wallet, VWExecuteParam calldata vweParam)
internal {
        address vwOwner = walletOwner[wallet];
        result[vwOwner][vweParam.code] = resCode;

        (uint256 dstChainId, uint256 srcChain, uint256 expTime) =
VWCode.chainidsAndExpTime(vweParam.code);
        require(dstChainId == block.chainid, 'E3');
        require(block.timestamp <= expTime, 'E6');
        require(domainSeparator[srcChain] != bytes32(0), 'E31');
        bytes32 rootHash = keccak256(
            abi.encode(
                APPROVE_SERVICE_TX_TYPEHASH,
                vweParam.code,
                keccak256(vweParam.data),
                vweParam.service,
                vweParam.gasToken,
                vweParam.gasTokenPrice,
                vweParam.priorityFee,
```

```
                vweParam.gasLimit,
                vweParam.isGateway
            )
        );
        if (vweParam.proof.length > 0) {
            rootHash = MerkleProof.processProof(vweParam.proof, rootHash);
            rootHash = keccak256(abi.encode(APPROVE_SERVICE_PROOF_TX_TYPEHASH,
rootHash));
        }
        // srcChain is the chain where user sign the rootHash
        if (Address.isContract(vwOwner)) {
            require(IWalletOwner(vwOwner).verifyVWParam(rootHash,
domainSeparator[srcChain], vweParam), 'E1');
        } else {
            SignLibrary.verify(vwOwner, domainSeparator[srcChain], rootHash,
vweParam.serviceSignature);
        }
        emit TxExecuted(wallet, vwOwner, vweParam.code, rootHash, resCode);
    }
```

Solution

It is recommended to add verification of feeReceiver and manager parameters.

Status

Acknowledged

[N7] [Suggestion] Redundant code

Category: Others

Content

The functions implemented in library OrderId are the same as those in library VWCode.

- contracts/libraries/OrderId.sol#L7-L17

```
 function genCode(
        uint128 nonce, uint32 time, uint32 srcChainId, uint32 dstChainId, uint16
oType, uint16 flag
    ) internal pure returns (uint code){
        code = (uint(nonce) << 128) + (uint(time) << 96) + (uint(srcChainId) << 64) +
(uint(dstChainId) << 32) + (uint(oType) << 16) + uint(flag);
    }

    function chainidsAndExpTime(uint code) internal pure returns (uint dstChainId,
```

```
uint srcChainId, uint time){
        dstChainId = (code >> 32) & ((1 << 32) - 1);
        srcChainId = (code >> 64) & ((1 << 32) - 1);
        time = (code >> 96) & ((1 << 32) - 1);
    }
```

Solution

It is recommended to delete redundant useless code according to the design of the business.

Status

Fixed

[N8] [Low] Reentrancy risks

Category: Reentrancy Vulnerability

Content

In the PayLock contract `punish` function, the `punishes` mapping is not used correctly, resulting in the risk of reentrancy.

- contracts/governance/PayLock.sol#L131-L159

```
    function punish(
        uint orderId,
        address node,
        address to,
        address[] calldata tokens,
        uint[] calldata amounts
    )
    external onlyOwner {
        require(tokens.length > 0 && tokens.length == amounts.length, "Invalid
length");
        for (uint i = 0; i < tokens.length; i++) {
            require(validTokens[tokens[i]], "INVALID_TOKEN");
            TokenBalance storage bal = nodeTokenBalance[node][tokens[i]];
            uint256 realAmount = amounts[i];
            if (amounts[i] > bal.numOnWithdraw) {
                if (bal.numTotal <= amounts[i].sub(uint(bal.numOnWithdraw))) {
                    realAmount = uint(bal.numTotal + bal.numOnWithdraw);
                    (bal.numTotal, bal.numOnWithdraw) = (0, 0);
                } else {
                    bal.numTotal -= (amounts[i] -
uint(bal.numOnWithdraw)).toUint128();
                    bal.numOnWithdraw = 0;
```

```
                }
            } else {
                bal.numOnWithdraw =
(uint(bal.numOnWithdraw).sub(amounts[i])).toUint128();
            }
            TransferHelper.safeTransfer2(tokens[i], to, realAmount);
        }
        punishes[node]++;
        emit NodePunished(orderId, amounts, tokens, node, to);
    }
```

Solution

It is recommended to modify the location of punishes[node]++ in the code so that it can correctly prevent re-entrancy attacks.

Status

Fixed

[N9] [High] Insufficient WithdrawPendingTime error

Category: Design Logic Audit

Content

In the PayLock contract configWithdrawPendingTime function, the withdrawPendingTime parameter should be set to greater than or equal to 7 days. Since configWithdrawPendingTime is associated with the order's term, configWithdrawPendingTime should be greater than the order's term.

- contracts/governance/PayLock.sol#L168-L172

```
    function configWithdrawPendingTime(uint period) external onlyOwner {
        require(period <= 7 days, "E27");
        withdrawPendingTime = period;
        emit WithdrawPendingTime(period);
    }
```

Solution

It is recommended to modify the time limit of withdrawPendingTime.

Status

Fixed

[N10] [High] Unverified Node Mortgage Requirement

Category: Design Logic Audit

Content

In the PayDB contract, when the user creates an order through the `createSrcOrder` function or `createSrcOrderETH` function, it is not verified whether the node's mortgage assets meet the mortgage requirements required for the created order.

Solution

It is recommended that when creating an order in the PayDB contract, it is linked to the data recording the node mortgage balance in the PayLock contract to verify whether the mortgage balance meets the mortgage requirements of the order.

Status

Fixed; The project team limit the `cparams[i].node` to their trusted nodes to avoid risk of insufficient collateral.

[N11] [Medium] Service nodes are at risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the PayDB contract, when the user creates an order through the createSrcOrder function or createSrcOrderETH function, he transfer assets to the `cparams[i].node`, and these nodes are at risk of rug-pull ,or private key leak.

Solution

Use of multi-signature technology to ensure that nodes can not do evil.

Status

Acknowledged; The project team stated that the nodes are verified and trusted by them.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002310070001	SlowMist Security Team	2023.09.25 - 2023.10.07	Medium 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, 3 medium risk, 5 low risk, 1 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist