Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.10.28, the SlowMist security team received the Orbiter Finance team's security audit application for OB ReturnCabin, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

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

The test method information:

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

The vulnerability severity level information:

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

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

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

3 Project Overview

3.1 Project Introduction

This is the OB ReturnCabin protocol of Orbiter Finance.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Regarding the status issue with durationCheck	Others	Information	Acknowledged

NO	Title	Category	Level	Status
N2	The issue of refund exceptions related to _makerFailed	Others	Critical	Fixed
N3	Regarding the withdrawal limit issue	Others	High	Fixed
N4	The value of _pledgeBalances is not being updated correctly	Others	Critical	Fixed
N5	Preemptive Initialization	Race Conditions Vulnerability	Suggestion	Acknowledged
N6	Missing event record	Malicious Event Log Audit	Suggestion	Acknowledged
N7	Suggestions on some redundant code	Others	Suggestion	Acknowledged
N8	Low-level call reminder	Others	Suggestion	Acknowledged
N9	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N10	Deflationary token compatibility issue	Others	Low	Acknowledged
N11	Regarding the verifyChallengeSource mechanism	Others	Information	Acknowledged
N12	withdrawVerification has an issue with replaying verification data	Others	Critical	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/Orbiter-Finance/OB_ReturnCabin/tree/main

commit: 0620520f56ea09afc363278e866b486751dd570c

review commit: fef25b972eed8a6af4fc6fbf6ef6f49ba153e737

Audit scope:

- contracts/ORExtraTransfer.sol
- contracts/ORFeeManager.sol
- contracts/ORMDCFactory.sol
- contracts/ORMakerDeposit.sol
- contracts/ORManager.sol
- contracts/ORSpvData.sol
- contracts/VersionAndEnableTime.sol
- contracts/example/ORChallengeSpvEra.sol
- contracts/example/ORChallengeSpvMainnet.sol
- contracts/example/OREventBinding.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:

ORExtraTransfer			
Function Name	Visibility	Mutability	Modifiers
transferNative	External	Payable	nonReentrant
transferErc20	External	Can Modify State	nonReentrant
batchTransferNative	External	Payable	nonReentrant
batchTransferErc20	External	Can Modify State	nonReentrant
batchTransferErc20WithTokens	External	Can Modify State	nonReentrant

ORFeeManager			
Function Name	Visibility	Mutability	Modifiers
durationCheck	Public	-	-
withdrawLockCheck	External	-	-
<Receive Ether>	External	Payable	-
<Constructor>	Public	Can Modify State	-
withdrawVerification	External	Can Modify State	nonReentrant
submit	External	Can Modify State	nonReentrant
updateDealer	External	Can Modify State	-
getDealerInfo	External	-	-
transferOwnership	Public	Can Modify State	onlyOwner
registerSubmitter	External	Can Modify State	onlyOwner

ORMDCFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
manager	External	-	-
implementation	External	-	-
mdcCreatedTotal	External	-	-
createMDC	External	Can Modify State	-
predictMDCAddress	External	-	-

ORMakerDeposit			
Function Name	Visibility	Mutability	Modifiers

ORMakerDeposit			
<Receive Ether>	External	Payable	-
initialize	External	Can Modify State	-
owner	External	-	-
mdcFactory	External	-	-
columnArrayHash	External	-	-
updateColumnArray	External	Can Modify State	onlyOwner
spv	External	-	-
updateSpvs	External	Can Modify State	onlyOwner
responseMakersHash	External	-	-
updateResponseMakers	External	Can Modify State	onlyOwner
freezeAssets	External	-	-
deposit	External	Payable	-
getWithdrawRequestList	External	-	-
withdrawRequest	External	Can Modify State	onlyOwner
withdraw	External	Can Modify State	onlyOwner
rulesRoot	External	-	-
updateRulesRoot	External	Payable	onlyOwner
updateRulesRootERC20	External	Can Modify State	onlyOwner
_updateRulesRoot	Private	Can Modify State	-
_addChallengeNode	Private	Can Modify State	-
canChallengeContinue	External	-	-
_canChallengeContinue	Private	-	-

ORMakerDeposit			
challenge	External	Payable	-
checkChallenge	External	Can Modify State	-
verifyChallengeSource	External	Can Modify State	-
verifyChallengeDest	External	Can Modify State	-
_challengerFailed	Internal	Can Modify State	-
_unFreezeToken	Internal	Can Modify State	-
_makerFailed	Internal	Can Modify State	-
_compareChallengerStatement	Internal	-	-

ORManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
registerChains	External	Can Modify State	onlyOwner
getPriorityFee	External	-	-
getChallengeGasUsed	External	-	-
getChallengeWithdrawDelay	External	-	-
updatePriorityFee	External	Can Modify State	onlyOwner
updateChallengeBasefee	External	Can Modify State	onlyOwner
updateChallengeWithdrawDelay	External	Can Modify State	onlyOwner
updateChainSpvs	External	Can Modify State	onlyOwner
getChainInfo	External	-	-
updateChainTokens	External	Can Modify State	onlyOwner
getChainTokenInfo	External	-	-

ORManager			
ebcIncludes	External	-	-
updateEbcs	External	Can Modify State	onlyOwner
submitter	External	-	-
updateSubmitter	External	Can Modify State	onlyOwner
protocolFee	External	-	-
updateProtocolFee	External	Can Modify State	onlyOwner
minChallengeRatio	External	-	-
updateMinChallengeRatio	External	Can Modify State	onlyOwner
challengeUserRatio	External	-	-
updateChallengeUserRatio	External	Can Modify State	onlyOwner
feeChallengeSecond	External	-	-
updateFeeChallengeSecond	External	Can Modify State	onlyOwner
feeTakeOnChallengeSecond	External	-	-
updateFeeTakeOnChallengeSecond	External	Can Modify State	onlyOwner
maxMDCLimit	External	-	-
updateMaxMDCLimit	External	Can Modify State	onlyOwner
spvDataContract	External	-	-
updateSpvDataContract	External	Can Modify State	onlyOwner
updateSpvBlockInterval	External	Can Modify State	onlyOwner
updateSpvDataInjectOwner	External	Can Modify State	onlyOwner
getExtraTransferContract	External	-	-
updateExtraTransferContracts	External	Can Modify State	onlyOwner

ORManager			
updateDecoderAddress	External	Can Modify State	onlyOwner
getRulesDecoder	External	-	-

ORSpvData			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
blockInterval	External	-	-
updateBlockInterval	External	Can Modify State	onlyManager
_calculateRoot	Internal	-	-
saveHistoryBlocksRoots	External	Can Modify State	-
getStartBlockNumber	External	-	-
injectOwner	External	-	-
updateInjectOwner	External	Can Modify State	onlyManager
injectBlocksRoots	External	Can Modify State	-

VersionAndEnableTime			
Function Name	Visibility	Mutability	Modifiers
versionIncreaseAndEnableTime	Internal	Can Modify State	-
getVersionAndEnableTime	External	-	-

ORChallengeSpvEra			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setSpvVerifierAddr	Public	Can Modify State	onlyOwner

ORChallengeSpvEra			
getSpvVerifierAddr	External	-	-
verifySourceTx	External	Can Modify State	-
verifyDestTx	External	Can Modify State	-
parseSourceTxProof	External	-	-
parseDestTxProof	External	-	-

ORChallengeSpvMainnet			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setSpvVerifierAddr	Public	Can Modify State	onlyOwner
getSpvVerifierAddr	External	-	-
verifySourceTx	External	Can Modify State	-
verifyDestTx	External	Can Modify State	-
parseSourceTxProof	External	-	-
parseDestTxProof	External	-	-

OREventBinding			
Function Name	Visibility	Mutability	Modifiers
getSecurityCode	Public	-	-
splitSecurityCode	Public	-	-
getAmountParams	Public	-	-
getResponseIntent	External	-	-
getResponseAmountFromIntent	External	-	-

4.3 Vulnerability Summary

[N1] [Information] Regarding the status issue with durationCheck

Category: Others

Content

The `durationCheck` function does not trigger an exception when the time exceeds one cycle. After exceeding one cycle, it transitions into the `FeeManagerDuration.withdraw` and `FeeManagerDuration.lock states`.

- contracts/ORFeeManager.sol

```
function durationCheck() public view returns (FeeMangerDuration duration) {
    uint256 challengeEnd = submissions.submitTimestamp +
ConstantsLib.DEALER_WITHDRAW_DELAY;
    if (block.timestamp <= challengeEnd) {
        return FeeMangerDuration.challenge;
    }

    uint256 mod = (block.timestamp - challengeEnd) % (ConstantsLib.WITHDRAW_DURATION +
ConstantsLib.LOCK_DURATION);
    if (mod <= ConstantsLib.WITHDRAW_DURATION) {
        return FeeMangerDuration.withdraw;
    } else {
        return FeeMangerDuration.lock;
    }
}
```

Solution

Verify whether the state calculation of durationCheck aligns with the design expectations. If it does not, an exception should be triggered once the end time is exceeded.

Status

Acknowledged

[N2] [Critical] The issue of refund exceptions related to _makerFailed

Category: Others

Content

After the challenge ends, all challengers except the winner receive refunds in Native Tokens, which can lead to irregular refund amounts. Users end up with Native Tokens that do not belong to them.

- contracts/ORMakerDeposit.sol

```
    function _makerFailed(
        ChallengeStatement memory challengeInfo,
        ChallengeStatement memory challengeInfoWinner,
        ChallengeResult memory result,
        address challenger,
        uint256 challengeIdentNum
    ) internal {
        if (result.winner == challenger) {
            uint256 userLostAmount = challengeInfo.freezeAmount1 >> 1;
            uint256 challengeUserAmount = (userLostAmount *
challengeInfo.challengeUserRatio) /
                ConstantsLib.RATIO_MULTIPLE;
            require(challengeUserAmount <= userLostAmount, "UAOF");
            challengeUserAmount += userLostAmount;

            uint256 challengerAmount = challengeInfo.freezeAmount0 +
challengeInfo.freezeAmount1 - challengeUserAmount;
            _challengeNodeList[challengeIdentNum].challengeFinished = true;


            address user = address(uint160(challengeInfo.sourceTxFrom));
            IERC20 token = IERC20(challengeInfo.freezeToken);

            if (challengeInfo.freezeToken == address(0)) {
                (bool sent1, ) = payable(user).call{value: challengeUserAmount}("");
                require(sent1, "ETH: SE1");

                (bool sent2, ) = payable(result.winner).call{
                    value: (challengerAmount +
                        challengeInfo.challengerVerifyTransactionFee +
                        ConstantsLib.MIN_CHALLENGE_DEPOSIT_AMOUNT)
                }("");
                require(sent2, "ETH: SE2");
            } else {
                token.safeTransfer(user, challengeUserAmount);
                token.safeTransfer(result.winner, challengerAmount);

                (bool sent3, ) = payable(result.winner).call{
                    value: ConstantsLib.MIN_CHALLENGE_DEPOSIT_AMOUNT +
                        challengeInfo.challengerVerifyTransactionFee +
                        challengeInfo.freezeAmount1
                }("");
```

```
                require(sent3, "ETH: SE3");
            }
        } else if (_compareChallengerStatement(challengeInfo, challengeInfoWinner) ==
true) {
            (bool sent4, ) = payable(challenger).call{
                value: ConstantsLib.MIN_CHALLENGE_DEPOSIT_AMOUNT +
challengeInfo.freezeAmount1
            }("");
            require(sent4, "ETH: SE4");
        }
    }
```

Solution

Refunds should be issued based on the token that the user used during the challenge.

Status

Fixed; The Erc20 Token logic has been disabled in version fef25b972eed8a6af4fc6fbf6ef6f49ba153e737.

[N3] [High] Regarding the withdrawal limit issue

Category: Others

Content

When withdrawing tokens other than the native token, the contract does not first deduct the frozen token amount.

This could potentially lead to irregularities in subsequent refund amounts.

- contracts/ORMakerDeposit.sol

```
function withdraw(address token) external onlyOwner {
    require(
        _withdrawRequestList[token].requestTimestamp > 0 &&
            block.timestamp >= _withdrawRequestList[token].requestTimestamp,
        "WTN"
    );
    WithdrawRequestList storage requestInfo = _withdrawRequestList[token];
    requestInfo.requestTimestamp = 0;

    uint256 balance;
    if (token == address(0)) {
        balance = address(this).balance - _freezeAssets[requestInfo.requestToken];

        require(balance >= requestInfo.requestAmount, "ETH: IF");

        (bool sent, ) = payable(msg.sender).call{value: requestInfo.requestAmount}
```

```
("");
        require(sent, "ETH: SE");
    } else {
        //@SlowMist
        IERC20(requestInfo.requestToken).safeTransfer(msg.sender,
requestInfo.requestAmount);
    }
}
```

Solution

The frozen token amount in the contract should be deducted.

Status

Fixed; The Erc20 Token logic has been disabled in version fef25b972eed8a6af4fc6fbf6ef6f49ba153e737.

[N4] [Critical] The value of _pledgeBalances is not being updated correctly

Category: Others

Content

The value of `_pledgeBalances[k]` is not updated when calculating the pledged ERC20 tokens, which can lead to inaccuracies in the pledged amount.

- contracts/ORMakerDeposit.sol

```
function updateRulesRootERC20(
    uint64 enableTime,
    address ebc,
    RuleLib.Rule[] calldata rules,
    RuleLib.RootWithVersion calldata rootWithVersion,
    uint64[] calldata sourceChainIds,
    uint256[] calldata pledgeAmounts,
    address token
) external onlyOwner {
    versionIncreaseAndEnableTime(enableTime);

    _updateRulesRoot(ebc, rules, rootWithVersion);

    require(sourceChainIds.length == pledgeAmounts.length, "SPL");

    for (uint256 i = 0; i < sourceChainIds.length; ) {
        bytes32 k = abi.encode(ebc, sourceChainIds[i], token).hash();

        if (pledgeAmounts[i] > _pledgeBalances[k]) {
```

```
            IERC20(token).safeTransferFrom(msg.sender, address(this), pledgeAmounts[i]
- _pledgeBalances[k]);
        }

        unchecked {
            i++;
        }
    }
}
```

Solution

After the transfer, the value of _pledgeBalances[k] should be updated correctly.

Status

Fixed; The Erc20 Token logic has been disabled in version fef25b972eed8a6af4fc6fbf6ef6f49ba153e737.

[N5] [Suggestion] Preemptive Initialization

Category: Race Conditions Vulnerability

Content

By calling the initialize function to initialize the contract, there is a potential issue that malicious attackers preemptively call the initialize function to initialize.

- contracts/ORMakerDeposit.sol

```
function initialize(address owner_) external {
    require(_owner == address(0), "_ONZ");
    require(owner_ != address(0), "OZ");

    _owner = owner_;
    _mdcFactory = IORMDCFactory(msg.sender);
}
```

Solution

It is suggested that the initialization operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Acknowledged

[N6] [Suggestion] Missing event record

Category: Malicious Event Log Audit

Content

The changes to the following key parameters have not been logged with corresponding events.

- contracts/ORManager.sol

```
function updatePriorityFee(uint64 priorityFee) external onlyOwner {
    _priorityFee = priorityFee;
}

function updateChallengeBasefee(uint32 challengeBasefee) external onlyOwner {
    _challengeBasefee = challengeBasefee;
}

function updateChallengeWithdrawDelay(uint32 challengeWithdrawDelay) external
onlyOwner {
    _challengeWithdrawDelay = challengeWithdrawDelay;
}

function updateSpvBlockInterval(uint64 spvBlockInterval_) external onlyOwner {
    IORSpvData(_spvDataContract).updateBlockInterval(spvBlockInterval_);
}

function updateSpvDataInjectOwner(address injectOwner_) external onlyOwner {
    IORSpvData(_spvDataContract).updateInjectOwner(injectOwner_);
}

function updateDecoderAddress(address decoderAddress) external onlyOwner {
    _decoderAddress = decoderAddress;
}
```

- contracts/example/ORChallengeSpvEra.sol

```
function setSpvVerifierAddr(address sourceTxVerifier, address destTxVerifier) public
onlyOwner {
    _sourceTxVerifier = sourceTxVerifier;
    _destTxVerifier = destTxVerifier;
}
```

- contracts/example/ORChallengeSpvMainnet.sol

```
    function setSpvVerifierAddr(address sourceTxVerifier, address destTxVerifier)
public onlyOwner {
        _sourceTxVerifier = sourceTxVerifier;
        _destTxVerifier = destTxVerifier;
    }
```

Solution

Record the corresponding event.

Status

Acknowledged

[N7] [Suggestion] Suggestions on some redundant code

Category: Others

Content

The `ext` parameter is unused and represents redundant code.

- contracts/ORExtraTransfer.sol

```
function transferNative(address payable to, bytes calldata ext) external payable
nonReentrant {
    ext;
    (bool success, ) = to.call{value: msg.value}("");
    require(success, "TF_ETH");
}

function batchTransferNative(
    address payable[] calldata tos,
    uint256[] calldata amounts,
    bytes[] calldata exts
) external payable nonReentrant {
    require(tos.length == amounts.length, "LF0");
    require(amounts.length == exts.length, "LF1");

    uint256 totalAmount = 0;
    for (uint256 i = 0; i < tos.length; i++) {
        totalAmount += amounts[i];

        (bool success, ) = tos[i].call{value: amounts[i]}("");
        require(success, "TF_ETH");
    }
```

```
    require(totalAmount == msg.value, "VI");
}
```

The status of `challengeStatus` is always `ChallengeStatus.none`, so the condition `require(challengeStatus == ChallengeStatus.none, 'WDC');` is always true, making it an ineffective check.

- contracts/ORFeeManager.sol

```
function withdrawVerification(
        MerkleTreeLib.SMTLeaf[] calldata smtLeaves
        bytes32[][] calldata siblings,
        uint8[] calldata startIndex,
        bytes32[] calldata firstZeroBits,
        uint256[] calldata bitmaps,
        uint256[] calldata withdrawAmount
    ) external nonReentrant {
        ...
        require(challengeStatus == ChallengeStatus.none, "WDC");
        ...
    }

    function submit(
        uint64 startBlock,
        uint64 endBlock,
        bytes32 profitRoot,
        bytes32 stateTransTreeRoot
    ) external override nonReentrant {
        ...
        require(challengeStatus == ChallengeStatus.none, "SDC");
        ...
    }
```

In the challenge function, the ruleKey parameter is unused.

- contracts/ORMakerDeposit.sol

Unused modifiers.

- contracts/ORFeeManager.sol

```
    modifier isChallengerQualified() {
        require(address(msg.sender).balance >=
address(IORManager(_manager).submitter()).balance, "NF");
```

```
        _;
    }
```

Solution

Remove this parameter if it's not needed.

Status

Acknowledged

[N8] [Suggestion] Low-level call reminder

Category: Others

Content

In the ORExtraTransfer contract, the contract uses low-level calls and does not limit the amount of gas used to transfer native tokens.

- contracts/ORExtraTransfer.sol

```
function transferNative(address payable to, bytes calldata ext) external payable
nonReentrant {
    ext;
    (bool success, ) = to.call{value: msg.value}("");
    require(success, "TF_ETH");
}

function batchTransferNative(
    address payable[] calldata tos,
    uint256[] calldata amounts,
    bytes[] calldata exts
) external payable nonReentrant {
    require(tos.length == amounts.length, "LF0");
    require(amounts.length == exts.length, "LF1");

    uint256 totalAmount = 0;
    for (uint256 i = 0; i < tos.length; i++) {
        totalAmount += amounts[i];

        (bool success, ) = tos[i].call{value: amounts[i]}("");
        require(success, "TF_ETH");
    }

    require(totalAmount == msg.value, "VI");
}
```

Solution

When using low-level calls, it is recommended to limit the amount of gas used.

Status

Acknowledged

[N9] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the protocol, the owner can set critical parameters. If the owner's private key is compromised, it could have serious consequences for the protocol.

- contracts/ORFeeManager.sol

```
owner can registerSubmitter
owner can transferOwnership
```

- contracts/ORMakerDeposit.sol

```
owner can updateColumnArray
owner can updateSpvs
owner can updateResponseMakers
owner can withdrawRequest
owner can withdraw
owner can updateRulesRoot
owner can updateRulesRootERC20
```

- contracts/ORFeeManager.sol

```
owner can registerChains
owner can updatePriorityFee
owner can updateChallengeBasefee
owner can updateChallengeWithdrawDelay
owner can updateChainSpvs
owner can updateChainTokens
owner can updateEbcs
owner can updateSubmitter
owner can updateProtocolFee
owner can updateMinChallengeRatio
owner can updateChallengeUserRatio
```

```
owner can updateFeeChallengeSecond
owner can updateFeeTakeOnChallengeSecond
owner can updateMaxMDCLimit
owner can updateSpvDataContract
owner can updateSpvBlockInterval
owner can updateSpvDataInjectOwner
owner can updateExtraTransferContracts
owner can updateDecoderAddress
```

- contracts/ORSpvData.sol

```
Manager can updateBlockInterval
Manager can updateInjectOwner
injectOwner can injectBlocksRoots
```

- contracts/example/ORChallengeSpvEra.sol

```
owner can setSpvVerifierAddr
```

- contracts/example/ORChallengeSpvMainnet.sol

```
owner can etSpvVerifierAddr
```

Solution

In the short term, during the early stages of the project, the protocol may need to frequently set various parameters to ensure the stable operation of the protocol. Therefore, transferring the ownership of core roles to a multisig management can effectively solve the single-point risk, but it cannot mitigate the excessive privilege risk. In the long run, after the protocol stabilizes, transferring the owner ownership to community governance and executing through a timelock can effectively mitigate the excessive privilege risk and increase the community users' trust in the protocol.

Status

Acknowledged; It is officially recommended that mdc deployers use at least multiple signatures to manage owner address.

[N10] [Low] Deflationary token compatibility issue

Category: Others

Content

The recorded freezeAmount1 does not reflect the actual number of tokens received. If the frozen token is deflationary, freezeAmount1 may exceed the actual tokens received. This could lead to irregular refund amounts for subsequent challengers.

- contracts/ORMakerDeposit.sol

```
function challenge(
    uint64 sourceTxTime,
    uint64 sourceChainId,
    uint64 sourceTxBlockNum,
    uint64 sourceTxIndex,
    bytes32 sourceTxHash,
    bytes32 ruleKey,
    address freezeToken,
    uint256 freezeAmount1,
    uint256 parentNodeNumOfTargetNode
) external payable {
    ...

    if (freezeToken == address(0)) {
        require(msg.value == (freezeAmount1 +
ConstantsLib.MIN_CHALLENGE_DEPOSIT_AMOUNT), "IF+MD");
        _freezeAssets[freezeToken] += freezeAmount0 + freezeAmount1 +
ConstantsLib.MIN_CHALLENGE_DEPOSIT_AMOUNT;
    } else {
        require(msg.value == ConstantsLib.MIN_CHALLENGE_DEPOSIT_AMOUNT, "IF");
        IERC20(freezeToken).safeTransferFrom(msg.sender, address(this),
freezeAmount1);
        _freezeAssets[freezeToken] += freezeAmount0 + freezeAmount1;
        _freezeAssets[address(0)] += ConstantsLib.MIN_CHALLENGE_DEPOSIT_AMOUNT;
    }

    ...

}
```

Solution

Base it on the actual amount received.

Status

Acknowledged; will not support deflationary tokens.

[N11] [Information] Regarding the verifyChallengeSource mechanism

Category: Others

Content

In `verifyChallengeSource`, the `winner` is determined by the submitter. If there are multiple challengers, the winner could be front-run. However, this aligns with the design intent, which allows any submitter to submit first.

- contracts/ORMakerDeposit.sol

```
function verifyChallengeSource(
    address challenger, //@SlowMist
    address spvAddress,
    uint64 sourceChainId,
    bytes calldata proof,
    bytes calldata rawDatas,
    bytes calldata encodeRuleBytes
) external override {
    uint256 startGasNum = gasleft();
    IORManager manager = IORManager(_mdcFactory.manager());
    require(manager.getChainInfo(sourceChainId).spvs.includes(spvAddress), "SPVI");
    IORChallengeSpv challengeSpv = IORChallengeSpv(spvAddress);
    require(challengeSpv.verifySourceTx(proof), "VF");
    PublicInputParseLib.PublicInputDataSource memory publicInputData =
challengeSpv.parseSourceTxProof(proof);
    for (uint256 i = 0; ; ) {
        require(

IORSpvData(manager.spvDataContract()).getStartBlockNumber(publicInputData.merkle_roots
[i]) != 0,
            "IBL"
        );

        if (i == publicInputData.merkle_roots.length - 1) {
            break;
        }

        unchecked {
            i++;
        }
    }
    require(
```

```
        (publicInputData.manage_contract_address == _mdcFactory.manager()) &&
            (publicInputData.mdc_contract_address == address(this)),
        "MCE"
    );
    require(publicInputData.chain_id == sourceChainId, "CID");
    bytes32 challengeId = abi.encode(publicInputData.chain_id,
publicInputData.tx_hash).hash();
    ChallengeStatement memory statement =
_challenges[challengeId].statement[challenger];
    require(statement.challengeTime > 0, "CTZ");
    require(_challenges[challengeId].result.verifiedTime0 == 0, "VT0NZ");
    ChallengeStatement storage statement_s =
_challenges[challengeId].statement[challenger];
    ChallengeResult storage result_s = _challenges[challengeId].result;
    result_s.verifiedTime0 = uint64(block.timestamp);

    require(publicInputData.time_stamp == statement.sourceTxTime, "ST");
    require(statement.sourceTxIndex == publicInputData.index, "TI");
    require(uint160(publicInputData.to) == uint160(_owner), "TNEO");
    require(
        statement.freezeToken == publicInputData.token &&
            publicInputData.token ==
publicInputData.manage_current_source_chain_mainnet_token &&
            publicInputData.manage_current_source_chain_mainnet_token ==
            publicInputData.manage_current_dest_chain_mainnet_token,
        "FTV"
    );
    require(statement.freezeAmount1 == publicInputData.amount << 1, "FALV");
    {
        uint timeDiff = block.timestamp - publicInputData.time_stamp;
        require(timeDiff >= publicInputData.min_verify_challenge_src_tx_second,
"MINTOF");
        require(timeDiff <= publicInputData.max_verify_challenge_src_tx_second,
"MAXTOF");
    }
    (address[] memory dealers, address[] memory ebcs, uint64[] memory chainIds,
address ebc) = abi.decode(
        rawDatas,
        (address[], address[], uint64[], address)
    );
    require(encodeRuleBytes.hash() == publicInputData.mdc_current_rule_value_hash,
"ERE");
    RuleLib.Rule memory rule =
IORRuleDecoder(IORManager(publicInputData.manage_contract_address).getRulesDecoder())
        .decodeRule(encodeRuleBytes);
    require(abi.encode(dealers, ebcs, chainIds).hash() ==
publicInputData.mdc_current_column_array_hash, "CHE");
    uint256 destChainId;
    {
```

```
        IOREventBinding.AmountParams memory ap =
IOREventBinding(ebc).getAmountParams(publicInputData.amount);
        require(ebc == ebcs[ap.ebcIndex - 1], "ENE");
        require(ap.chainIdIndex <= chainIds.length, "COF");
        destChainId = chainIds[ap.chainIdIndex - 1];
        require(
            uint160(statement.freezeToken) ==
uint160(publicInputData.manage_current_dest_chain_mainnet_token),
            "DTV"
        );
    }

    RuleLib.RuleOneway memory ro = RuleLib.convertToOneway(rule,
publicInputData.chain_id);
    uint256 destAmount = IOREventBinding(ebc).getResponseAmountFromIntent(
        IOREventBinding(ebc).getResponseIntent(publicInputData.amount, ro)
    );
    require(destChainId == ro.destChainId, "DCI");

    {

        uint256 slot = uint256(abi.encode(ebc, 6).hash());
        require(slot == publicInputData.mdc_rule_root_slot, "RRSE");
    }

    statement_s.sourceTxFrom = publicInputData.from;

    statement_s.challengeUserRatio =
publicInputData.manage_current_challenge_user_ratio;

    result_s.winner = challenger; // @SlowMist


    // [minVerifyChallengeDestTxSecond, maxVerifyChallengeDestTxSecond, nonce,
destChainId, destAddress, destToken, destAmount, responeMakersHash]
    result_s.verifiedDataHash0 = abi
        .encode(
            verifiedDataInfo({
                minChallengeSecond:
publicInputData.min_verify_challenge_dest_tx_second,
                maxChallengeSecond:
publicInputData.max_verify_challenge_dest_tx_second,
                nonce: publicInputData.nonce,
                destChainId: destChainId,
                from: publicInputData.from,
                destToken: ro.destToken,
                destAmount: destAmount,
                responseMakersHash: publicInputData.mdc_current_response_makers_hash,
                responseTime: ro.responseTime
```

```
            })
        )
        .hash();
    emit ChallengeInfoUpdated({challengeId: challengeId, statement: statement_s,
result: result_s});
    uint128 actualGasPrice = uint128(block.basefee +
IORManager(_mdcFactory.manager()).getPriorityFee());
    statement_s.challengerVerifyTransactionFee +=
        uint128((startGasNum - gasleft() +
IORManager(_mdcFactory.manager()).getChallengeGasUsed())) *
        actualGasPrice;
}
```

Solution

Status

Acknowledged

[N12] [Critical] withdrawVerification has an issue with replaying verification data

Category: Others

Content

In withdrawVerification, the key value is calculated using smtLeaves[0] instead of smtLeaves[i]. This means that if a user has multiple withdrawal entries, they could replay the transaction to claim rewards multiple times.

- contracts/ORFeeManager.sol

```
function withdrawVerification(
    MerkleTreeLib.SMTLeaf[] calldata smtLeaves,
    bytes32[][] calldata siblings,
    uint8[] calldata startIndex,
    bytes32[] calldata firstZeroBits,
    uint256[] calldata bitmaps,
    uint256[] calldata withdrawAmount
) external nonReentrant {
    require(durationCheck() == FeeMangerDuration.withdraw, "WE");
    require(challengeStatus == ChallengeStatus.none, "WDC");
    for (uint256 i = 0; i < smtLeaves.length; ) {
        bytes32 key = keccak256(abi.encode(smtLeaves[0])); //@SlowMist
        require(withdrawLock[msg.sender][key] < submissions.submitTimestamp, "WL");
        withdrawLock[msg.sender][key] = submissions.submitTimestamp;

        address token = smtLeaves[i].token;
        address user = smtLeaves[i].user;
```

```
        uint64 chainId = smtLeaves[i].chainId;
        uint256 debt = smtLeaves[i].debt;
        uint256 amount = smtLeaves[i].amount;

        require(msg.sender == user, "NU");
        require(withdrawAmount[i] <= amount, "UIF");

        require(
            keccak256(abi.encode(chainId, token, user)).verify(
                keccak256(abi.encode(token, chainId, amount, debt)),
                bitmaps[i],
                submissions.profitRoot,
                firstZeroBits[i],
                startIndex[i],
                siblings[i]
            ),
            "merkle root verify failed"
        );

        if (token != address(0)) {
            IERC20(token).safeTransfer(msg.sender, withdrawAmount[i]);
        } else {
            (bool success, ) = payable(msg.sender).call{value: withdrawAmount[i]}
("");
            require(success, "ETH: IF");
        }
        emit Withdraw(msg.sender, chainId, token, debt, withdrawAmount[i]);
        unchecked {
            i += 1;
        }
    }
}
```

Solution

smtLeaves[i] should be used as the key, rather than smtLeaves[0].

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002411040001	SlowMist Security Team	2024.10.28 - 2024.11.04	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 3 critical risk, 1 high risk, 1 medium risk, 1 low risk, 4 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