Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.05.06, the SlowMist security team received the Brevis Network team's security audit application for Incentra Contracts, 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

The Incentra reward system is a blockchain-based cross-chain incentive platform that allows project parties to create various activities, distribute token rewards based on user behavior, and support cross-chain verification and claiming.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing duplicate check for topRoot in epoch start may cause	Design Logic Audit	Information	Acknowledged

NO	Title	Category	Level	Status
	overwrite of unsent messages			
N2	Preemptive Initialization	Race Conditions Vulnerability	Suggestion	Acknowledged
N3	Owner zero address check recommendation	Others	Suggestion	Fixed
N4	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N5	Missing zero address check	Others	Suggestion	Acknowledged
N6	Regarding the issue of replaying the sendTopRoot message	Others	Information	Confirming
N7	Missing event record	Malicious Event Log Audit	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

https://github.com/brevis-network/incentra-contracts

Audit Commit: c2b0ec0ef085b3fe1c078b9ec9a4cc11a61a98d3

Review Commit: 95b261f73c70b38f33f16ae9e7f23149ad770017

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:

AccessControl			
Function Name	Visibility	Mutability	Modifiers

AccessControl			
hasRole	Public	-	-
numRoleAccounts	Public	-	-
getRoleAccounts	Public	-	-
grantRole	Public	Can Modify State	onlyOwner
grantRoles	Public	Can Modify State	onlyOwner
revokeRole	Public	Can Modify State	onlyOwner
revokeRoles	Public	Can Modify State	onlyOwner
renounceRole	Public	Can Modify State	-
_grantRole	Internal	Can Modify State	-
_revokeRole	Internal	Can Modify State	-

Ownable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initOwner	Internal	Can Modify State	-
initOwner	Internal	Can Modify State	-
owner	Public	-	-
transferOwnership	Public	Can Modify State	onlyOwner
_setOwner	Private	Can Modify State	-

BrevisProofApp			
Function Name	Visibility	Mutability	Modifiers
_checkBrevisProof	Internal	Can Modify State	-

BrevisProofApp			
_checkBrevisAggProof	Internal	Can Modify State	-

BrevisProofRelay			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
submitProof	External	Can Modify State	onlyRole
validateProofAppData	External	-	-
submitAggProof	External	Can Modify State	onlyRole
validateAggProofData	External	-	-
addCampaigns	External	Can Modify State	onlyRole
removeCampaigns	External	Can Modify State	onlyRole
addAdmin	External	Can Modify State	onlyOwner
removeAdmin	External	Can Modify State	onlyOwner
setBrevisProof	External	Can Modify State	onlyOwner

CampaignCL			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
refund	External	Can Modify State	-
_useEnumerableMap	Internal	-	-

RewardsSubmissionCL			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-

RewardsSubmissionCL			
_useEnumerableMap	Internal	-	-

RewardsUpdateCL			
Function Name	Visibility	Mutability	Modifiers
_initConfig	Internal	Can Modify State	-
updateTotalFee	External	Can Modify State	onlyRole
_getDataChainId	Internal	-	-
_getHeaderSize	Internal	-	-
_getSizePerEarner	Internal	-	-
_updateRewards	Internal	Can Modify State	-
_addDirectRewards	Internal	Can Modify State	-
_addIndirectRewards	Internal	Can Modify State	-

CampaignGeneric			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
refund	External	Can Modify State	-
_useEnumerableMap	Internal	-	-

RewardsSubmissionGeneric			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
_useEnumerableMap	Internal	-	-

RewardsUpdateGeneric			
Function Name	Visibility	Mutability	Modifiers
_initConfig	Internal	Can Modify State	-
_getDataChainId	Internal	-	-
_getHeaderSize	Internal	-	-
_getSizePerEarner	Internal	-	-
_updateRewards	Internal	Can Modify State	-
_addRewards	Internal	Can Modify State	-
checkExtraData	Private	-	-

TotalFee			
Function Name	Visibility	Mutability	Modifiers
_updateFee	Internal	Can Modify State	-

MessageSenderApp			
Function Name	Visibility	Mutability	Modifiers
sendMessage	Internal	Can Modify State	-

MessageReceiverApp			
Function Name	Visibility	Mutability	Modifiers
executeMessage	External	Can Modify State	onlyMessageBus
_executeMessage	Internal	Can Modify State	-

CampaignRewardsClaim			
Function Name	Visibility	Mutability	Modifiers

CampaignRewardsClaim			
init	External	Can Modify State	-
refund	External	Can Modify State	-
claim	External	Can Modify State	-
claimWithRecipient	External	Can Modify State	-
updateRoot	External	Can Modify State	onlyRole
_executeMessage	Internal	Can Modify State	-
getTokens	Public	-	-
viewClaimedRewards	External	-	-
setSubmissionContract	External	Can Modify State	onlyOwner
setMessageBus	External	Can Modify State	onlyOwner
setGracePeriod	External	Can Modify State	onlyOwner
_claim	Private	Can Modify State	-
_updateRoot	Private	Can Modify State	-
verifyMerkleProof	Private	-	-

RewardsMerkle			
Function Name	Visibility	Mutability	Modifiers
startEpoch	External	Can Modify State	onlyRole
startSubRootGen	External	Can Modify State	onlyRole
genSubRoot	External	Can Modify State	-
genTopRoot	Public	Can Modify State	-
sendTopRoot	Public	Payable	-
genAndSendTopRoot	External	Payable	-

RewardsMerkle			
getMerkleProof	External	-	-
setMessageBus	External	Can Modify State	onlyOwner
genMerkleRoot	Private	-	-
genMerkleProof	Private	-	-

RewardsClaim			
Function Name	Visibility	Mutability	Modifiers
claim	External	Can Modify State	-
claimWithRecipient	External	Can Modify State	-
viewUnclaimedRewards	External	-	-
setGracePeriod	External	Can Modify State	onlyOwner
_claim	Internal	Can Modify State	-

ClaimAll			
Function Name	Visibility	Mutability	Modifiers
claimAll	Public	Can Modify State	-
claimAll	Public	Can Modify State	-
claimAll	External	Can Modify State	-

RewardsStorage			
Function Name	Visibility	Mutability	Modifiers
setVk	External	Can Modify State	onlyOwner
getTokens	Public	-	-
getRewardAmount	Public	-	-

RewardsStorage			
viewTotalRewards	External	-	-
getRewardsLength	External	-	-
updateRewards	External	Can Modify State	onlyRole
_initTokens	Internal	Can Modify State	-
_checkProofAndGetAppId	Internal	Can Modify State	-
_useEnumerableMap	Internal	-	-
_getDataChainId	Internal	-	-
_getHeaderSize	Internal	-	-
_getSizePerEarner	Internal	-	-
_updateRewards	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Information] Missing duplicate check for topRoot in epoch start may cause overwrite of unsent messages

Category: Design Logic Audit

Content

During the start Epoch, there was no check to see if the `topRoot` generated this time had already been sent, which could potentially lead to mistakenly overwriting unsent `topRoot` messages.

- src/rewards/cross-chain/RewardsMerkle.sol

```
function startEpoch(uint64 epoch) external onlyRole(REWARD_UPDATER_ROLE) {
        require(state == State.Idle, "invalid state");
        require(epoch > currEpoch, "invalid epoch");
        currEpoch = epoch;
        state = State.RewardsSubmission;
        subRoots.clear();
    }
```

```
function genTopRoot(uint64 epoch) public {
    require(state == State.TopRootGeneration, "invalid state");
    require(epoch == currEpoch, "invalid epoch");
    topRoot = genMerkleRoot(subRoots.values());
    state = State.Idle;
    emit TopRootGenerated(currEpoch, topRoot);
}
function sendTopRoot(address _receiver, uint64 _dstChainId) public payable {
    require(messageBus != address(0), "message bus not set");
    require(state == State.Idle, "invalid state");
    require(topRoot != bytes32(0), "top root not generated");
    bytes memory message = abi.encode(currEpoch, topRoot);
    sendMessage(_receiver, _dstChainId, message, msg.value);
    emit TopRootSent(currEpoch, topRoot, _receiver, _dstChainId);
}
```

Solution

Add a sending status, only when the message is confirmed to be sent can the next epoch begin.

Status

Acknowledged; Project party: This is in line with the design expectations, it can be corrected by waiting for the next epoch's root update.

[N2] [Suggestion] Preemptive Initialization

Category: Race Conditions Vulnerability

Content

The `init` function does not restrict the caller, so there is a risk of being called ahead of time.

- src/concentrated-liquidity/CampaignCL.sol

```
    function init(
        ConfigCL calldata cfg,
        IBrevisProof brv,
        address owner,
        bytes32[] calldata vks,
        uint64 dataChainId,
        address rewardUpdater
    ) external {
        initOwner(owner);
        _initConfig(cfg, brv, vks, dataChainId);
        grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
    }
```

- src/concentrated-liquidity/RewardsSubmissionCL.sol

```
function init(
    ConfigCL calldata cfg,
    IBrevisProof brv,
    address owner,
    bytes32[] calldata vks,
    uint64 dataChainId,
    address rewardUpdater
) external {
    initOwner(owner);
    _initConfig(cfg, brv, vks, dataChainId);
    grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
}
```

- src/generic/CampaignGeneric.sol

```
function init(
    ConfigGeneric calldata cfg,
    IBrevisProof brv,
    address owner,
    bytes32[] calldata vks,
    uint64 dataChainId,
    address rewardUpdater
) external {
    initOwner(owner);
    _initConfig(cfg, brv, vks, dataChainId);
    grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
}
```

- src/generic/RewardsSubmissionGeneric.sol

```
function init(
    ConfigGeneric calldata cfg,
    IBrevisProof brv,
    address owner,
    bytes32[] calldata vks,
    uint64 dataChainId,
    address rewardUpdater
) external {
    initOwner(owner);
    _initConfig(cfg, brv, vks, dataChainId);
    grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
}
```

- src/rewards/cross-chain/CampaignRewardsClaim.sol

```
    function init(
        Config calldata cfg,
        address owner,
        address root_updater,
        address _messageBus,
        uint64 _submissionChainId,
        address _submissionAddress
    ) external {
        initOwner(owner);
        grantRole(ROOT_UPDATER_ROLE, root_updater);
        config = cfg;
        messageBus = _messageBus;
        submissionChainId = _submissionChainId;
        submissionAddress = _submissionAddress;
    }
```

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

[N3] [Suggestion] Owner zero address check recommendation

Category: Others

Content

If the `owner` set is the 0 address, there is a possibility of init being initialized for the second time.

- src/concentrated-liquidity/CampaignCL.sol

```
    function init(
        ConfigCL calldata cfg,
        IBrevisProof brv,
        address owner,
        bytes32[] calldata vks,
        uint64 dataChainId,
        address rewardUpdater
    ) external {
```

```
        initOwner(owner);
        _initConfig(cfg, brv, vks, dataChainId);
        grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
    }
```

- src/concentrated-liquidity/RewardsSubmissionCL.sol

```
    function init(
        ConfigCL calldata cfg,
        IBrevisProof brv,
        address owner,
        bytes32[] calldata vks,
        uint64 dataChainId,
        address rewardUpdater
    ) external {
        initOwner(owner);
        _initConfig(cfg, brv, vks, dataChainId);
        grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
    }
```

- src/generic/CampaignGeneric.sol

```
    function init(
        ConfigGeneric calldata cfg,
        IBrevisProof brv,
        address owner,
        bytes32[] calldata vks,
        uint64 dataChainId,
        address rewardUpdater
    ) external {
        initOwner(owner);
        _initConfig(cfg, brv, vks, dataChainId);
        grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
    }
```

- src/generic/RewardsSubmissionGeneric.sol

```
    function init(
        ConfigGeneric calldata cfg,
        IBrevisProof brv,
        address owner,
        bytes32[] calldata vks,
        uint64 dataChainId,
        address rewardUpdater
    ) external {
```

```
        initOwner(owner);
        _initConfig(cfg, brv, vks, dataChainId);
        grantRole(REWARD_UPDATER_ROLE, rewardUpdater);
    }
```

- src/rewards/cross-chain/CampaignRewardsClaim.sol

```
    function init(
        Config calldata cfg,
        address owner,
        address root_updater,
        address _messageBus,
        uint64 _submissionChainId,
        address _submissionAddress
    ) external {
        initOwner(owner);
        grantRole(ROOT_UPDATER_ROLE, root_updater);
        config = cfg;
        messageBus = _messageBus;
        submissionChainId = _submissionChainId;
        submissionAddress = _submissionAddress;
    }
```

Solution

Restricting the owner from being the zero address.

Status

Fixed

[N4] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

The Owner is the most privileged role in the system, with full control over key parameters and the authority to manage all other roles; if the Owner's private key is compromised, it can lead to a complete takeover and severe asset losses. Other roles, such as Admin, Campaign, and Reward Updater roles, are granted limited permissions for specific operations like campaign management, proof submission, and reward updating, but do not have overarching control of the system.

- src/access/AccessControl.sol

```
Owner can grantRole
Owner can grantRoles
Owner can revokeRole
Owner can revokeRoles
```

- src/access/Ownable.sol

```
Owner can transferOwnership
```

- src/brevis/BrevisProofRelay.sol

```
CAMPAIGN_ROLE can submitProof
CAMPAIGN_ROLE can submitAggProof
ADMIN_ROLE can addCampaigns
ADMIN_ROLE can removeCampaigns
Owner can addAdmin
Owner can removeAdmin
Owner can setBrevisProof
```

- src/concentrated-liquidity/RewardsUpdateCL.sol

```
REWARD_UPDATER_ROLE can updateTotalFee
```

- src/concentrated-liquidity/RewardsUpdateCL.sol

```
REWARD_UPDATER_ROLE can updateTotalFee
```

- src/rewards/cross-chain/CampaignRewardsClaim.sol

```
ROOT_UPDATER_ROLE can updateRoot
Owner can setSubmissionContract
Owner can setMessageBus
Owenr can setGracePeriod
```

- src/rewards/cross-chain/RewardsMerkle.sol

```
REWARD_UPDATER_ROLE can startEpoch
REWARD_UPDATER_ROLE can startSubRootGen
```

```
Owner can setMessageBus
```

- src/rewards/same-chain/RewardsClaim.sol

```
Owner can setGracePeriod
```

- src/rewards/RewardsStorage.sol

```
Owner can setVk
REWARD_UPDATER_ROLE can updateRewards
```

Solution

In the short term, transferring privileged roles to a multi-signature wallet can effectively mitigate the single point of failure risk. In the long term, transferring privileged roles to DAO governance can effectively address the risk of excessive privilege. During the transition period, managing through multi-signature with delayed transaction execution via timelock can effectively mitigate the risk of excessive privilege.

Status

Acknowledged; Project party: In the early stage, multi-signature will be used to manage the owner role, and later the role will be transferred to the DAO.

[N5] [Suggestion] Missing zero address check

Category: Others

Content

It is recommended to add zero address (address(0)) checks to the setMessageBus, setSubmissionContract, and setBrevisProof functions to prevent accidentally setting critical contract addresses to an invalid value. Assigning a zero address may lead to loss of functionality or security issues. Always validate input addresses before updating state variables.

- src/rewards/cross-chain/CampaignRewardsClaim.sol

```
function setMessageBus(address _messageBus) external onlyOwner {
    messageBus = _messageBus;
    emit MessageBusUpdated(_messageBus);
}
```

```
function setSubmissionContract(uint64 _submissionChainId, address _submissionAddress)
external onlyOwner {
    submissionChainId = _submissionChainId;
    submissionAddress = _submissionAddress;
    emit SubmissionContractUpdated(_submissionChainId, _submissionAddress);
}
```

- src/brevis/BrevisProofRelay.sol

```
function setBrevisProof(address _brevisProof) external onlyOwner {
    address oldAddr = address(brevisProof);

    brevisProof = IBrevisProof(_brevisProof);
    emit BrevisProofUpdated(oldAddr, _brevisProof);
}
```

Solution

Status

Acknowledged

[N6] [Information] Regarding the issue of replaying the sendTopRoot message

Category: Others

Content

The calculated top Root can be called by anyone and the recipient can be any address. msg.value can be 0, which poses risks.

- src/rewards/cross-chain/RewardsMerkle.sol

```
function sendTopRoot(address _receiver, uint64 _dstChainId) public payable {
    require(messageBus != address(0), "message bus not set");
    require(state == State.Idle, "invalid state");
    require(topRoot != bytes32(0), "top root not generated");
    bytes memory message = abi.encode(currEpoch, topRoot);
    sendMessage(_receiver, _dstChainId, message, msg.value);
    emit TopRootSent(currEpoch, topRoot, _receiver, _dstChainId);
}
```

Solution

Need to check the recipient, verify the amount matches the expectation, and ensure the message cannot be replayed.

Status

Confirming; Project party: The Celer cBridge MessageBus contract includes built-in replay protection, ensuring that the same message cannot be processed more than once. And it can only be executed successfully on the correct chain.

[N7] [Suggestion] Missing event record

Category: Malicious Event Log Audit

Content

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

- src/rewards/cross-chain/RewardsMerkle.sol

```
function startSubRootGen(uint64 epoch) external onlyRole(REWARD_UPDATER_ROLE) {
    require(state == State.RewardsSubmission, "invalid state");
    require(currEpoch == epoch, "invalid epoch");
    state = State.SubRootsGeneration;
}
```

Solution

Record the corresponding event.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002505090001	SlowMist Security Team	2025.05.06 - 2025.05.09	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 1 medium 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