Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.04.14, the SlowMist security team received the team's security audit application for SGNv2 Message Apps, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

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

The test method information:

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

The vulnerability severity level information:

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

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

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

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

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

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

Serial Number	Audit Class	Audit Subclass
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
		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

Audit version:

https://github.com/celer-network/sgn-v2-contracts/tree/main/contracts/message/apps

commit: ae33d2f935519598047f9c41e84a6159fd704578

Audit scope:

NFTMCN.sol

NFTPeg.sol

NFTBridge.sol

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirming
N2	Missing event records	Others	Suggestion	Confirming
N3	Lack of black hole address judgment	Others	Suggestion	Confirming
N4	Coding specification optimization	Others	Suggestion	Confirming

4 Code Overview

4.1 Contracts Description

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:

MCNNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721
bridgeMint	External	Can Modify State	onlyNftBridge
totalFee	External	-	-
crossChain	External	Payable	-
mint	External	Can Modify State	onlyOwner
setNFTBridge	Public	Can Modify State	onlyOwner

PegNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721
bridgeMint	External	Can Modify State	onlyNftBridge
burn	External	Can Modify State	onlyNftBridge

NFTBridge			
Function Name	Visibility	Mutability	Modifiers

NFTBridge			
<Constructor>	Public	Can Modify State	-
init	External	Can Modify State	-
totalFee	External	-	-
deposit	External	Payable	-
burn	External	Payable	-
sendMsg	External	Payable	-
executeMessage	External	Payable	onlyMessageBus
checkAddr	Internal	-	-
msgBus	Internal	Can Modify State	-
setDestNFT	External	Can Modify State	onlyOwner
setDestNFTs	External	Can Modify State	onlyOwner
setTxFee	External	Can Modify State	onlyOwner
setDestBridge	External	Can Modify State	onlyOwner
setDestBridges	External	Can Modify State	onlyOwner
claimFee	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

1.In the NFTPeg contract, the nftBridge role can mint NFT arbitrarily through the bridgeMint function and burn any users NFT through the burn function.

Code location:

NFTPeg.sol#22-33

```
    function bridgeMint(
        address to,
        uint256 id,
        string memory uri
    ) external onlyNftBridge {
        _mint(to, id);
        _setTokenURI(id, uri);
    }

    function burn(uint256 id) external onlyNftBridge {
        _burn(id);
    }
```

2.In the NFTMCN contract, the owner role can mint NFT arbitrarily through the mint function and the owner role can add a new nftBridge role through the setNFTBridge function, and the nftBridge role can mint NFT arbitrarily through the bridgeMint function.

Code location:

NFTMCN.sol#40-47,67-74

```
    function bridgeMint(
        address to,
        uint256 id,
        string memory uri
    ) external onlyNftBridge {
        _mint(to, id);
        _setTokenURI(id, uri);
    }

    function mint(
        address to,
        uint256 id,
        string memory uri
    ) external onlyOwner {
        _mint(to, id);
        _setTokenURI(id, uri);
    }
```

3.In the NFTBridge contract, the owner role can change the destTxFee through the setTxFee function, and the destTxFee doesn't set a range limit and the setTxFee function has no event log.

Code location:

NFTBridge.sol#215-217

```
    function setTxFee(uint64 chid, uint256 fee) external onlyOwner {
        destTxFee[chid] = fee;
    }
```

Solution

1. It is recommended to use a time lock mechanism or community governance to restrict.
2. It is recommended to use a time lock mechanism or community governance to restrict.
3. It is recommended to use a time lock mechanism or community governance to restrict and set a scope limit to the destTxFee, and record events when sensitive parameters are modified for subsequent self-inspection or community review.

Status

Confirming

[N2] [Suggestion] Missing event records

Category: Others

Content

In the NFTBridge contract, the owner role can set the dstNft addresses, change the destFee and dstNftBridge addresses through the setDestNFT, setDestNFTs, setTxFee, setDestBridge, and setDestBridges functions, but there are no no events logging performed.

Code location:

NFTBridge.sol#195-229

```
    function setDestNFT(
        address srcNft,
        uint64 dstChid,
        address dstNft
    ) external onlyOwner {
        destNFTAddr[srcNft][dstChid] = dstNft;
    }

    // set all dest chains
    function setDestNFTs(
        address srcNft,
        uint64[] calldata dstChid,
        address[] calldata dstNft
    ) external onlyOwner {
        for (uint256 i = 0; i < dstChid.length; i++) {
            destNFTAddr[srcNft][dstChid[i]] = dstNft[i];
        }
    }

    // set destTxFee
    function setTxFee(uint64 chid, uint256 fee) external onlyOwner {
        destTxFee[chid] = fee;
    }

    // set per chain id, nft bridge address
    function setDestBridge(uint64 dstChid, address dstNftBridge) external onlyOwner {
        destBridge[dstChid] = dstNftBridge;
    }

    // batch set nft bridge addresses for multiple chainids
    function setDestBridges(uint64[] calldata dstChid, address[] calldata
dstNftBridge) external onlyOwner {
        for (uint256 i = 0; i < dstChid.length; i++) {
            destBridge[dstChid[i]] = dstNftBridge[i];
        }
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for subsequent self-inspection or community review.

Status

Confirming

[N3] [Suggestion] Lack of black hole address judgment

Category: Others

Content

1.In the NFTMCN contract, the owner role can change the nftBridge address through the setNFTBridge function, and there is a lack of judgment on whether to set it as a black hole address.

Code location:

NFTMCN.sol#76-79

```
    function setNFTBridge(address _newBridge) public onlyOwner {
        nftBridge = _newBridge;
        emit NFTBridgeUpdated(_newBridge);
    }
```

2.In the NFTBridge contract, the owner role can set the dstNft addresses and dstNftBridge addresses through the setDestNFT, setDestNFTs, setDestBridge, and setDestBridges functions, and there are ack of judgment on whether to set it as a black hole address.

Code location:

NFTBridge.sol#195-212, 220-229

```
   function setDestNFT(
        address srcNft,
        uint64 dstChid,
        address dstNft
    ) external onlyOwner {
        destNFTAddr[srcNft][dstChid] = dstNft;
    }

    // set all dest chains
    function setDestNFTs(
        address srcNft,
        uint64[] calldata dstChid,
        address[] calldata dstNft
    ) external onlyOwner {
        for (uint256 i = 0; i < dstChid.length; i++) {
            destNFTAddr[srcNft][dstChid[i]] = dstNft[i];
        }
    }

    function setDestBridge(uint64 dstChid, address dstNftBridge) external onlyOwner {
        destBridge[dstChid] = dstNftBridge;
    }

    // batch set nft bridge addresses for multiple chainids
    function setDestBridges(uint64[] calldata dstChid, address[] calldata
dstNftBridge) external onlyOwner {
        for (uint256 i = 0; i < dstChid.length; i++) {
            destBridge[dstChid[i]] = dstNftBridge[i];
        }
    }
```

Solution

It is recommended to add the judgment of the black hole address.

require(_addr != address(0));

Status

Confirming

[N4] [Suggestion] Coding specification optimization

Category: Others

Content

According to the coding standard, all function names of internally called functions need to be named with _ at the beginning.

Code location:

NFTBridge.sol#175-191

```
    function checkAddr(address _nft, uint64 _dstChid) internal view returns (address
dstBridge, address dstNft) {
        dstBridge = destBridge[_dstChid];
        require(dstBridge != address(0), "dest NFT Bridge not found");
        dstNft = destNFTAddr[_nft][_dstChid];
        require(dstNft != address(0), "dest NFT not found");
    }

    // check fee and call msgbus sendMessage
    function msgBus(
        address _dstBridge,
        uint64 _dstChid,
        bytes memory message
    ) internal {
        uint256 fee = IMessageBus(messageBus).calcFee(message);
        require(msg.value >= fee + destTxFee[_dstChid], "insufficient fee");
        IMessageBus(messageBus).sendMessage{value: fee}(_dstBridge, _dstChid,
message);
    }
```

Solution

It is recommended to start with _ to name the internally called function.

Status

Confirming

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002204180001	SlowMist Security Team	2022.04.14 - 2022.04.18	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, 3 suggestion vulnerabilities. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.