Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.05.13, the SlowMist security team received the Bitlayer team's security audit application for bitlayer-bridge Phase 2, 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 BitlayerBridgeV2 contract is used to manage the cross-chain functionality of RUNES & BRC20 tokens on the Bitlayer network.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	The validity of the token contract	Design Logic Audit	High	Fixed

NO	Title	Category	Level	Status
	address is not checked			
N2	Unlimited huge amount minting	Design Logic Audit	Medium	Acknowledged
N3	The returned leftQuota value is inaccurate	Design Logic Audit	Low	Fixed
N4	Missing zero address check	Others	Suggestion	Fixed
N5	Unchecked parameter value is not 0	Others	Suggestion	Fixed
N6	Redundant code	Others	Suggestion	Acknowledged
N7	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/bitlayer-org/bitlayer-bridge

Initial audit commit: 875321eacee8f8e92396d3a834a9767f3bdd37b8

Final audit commit: d8b36e26f9cc8ea4359cf437d0f3426cf703891c

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:

BitlayerBridgeV2			
Function Name	Visibility	Mutability	Modifiers

BitlayerBridgeV2			
version	External	-	-
setRoles	External	Can Modify State	onlyRole
initToken	External	Can Modify State	onlyRole
setPeriodMintLimit	Public	Can Modify State	onlyRole
setMintSplitLine	Public	Can Modify State	onlyRole
setCrossoutFee	Public	Can Modify State	onlyRole
setPeriodInterval	Public	Can Modify State	onlyRole
startNewPeriod	External	Can Modify State	onlyRole
directMint	External	Can Modify State	onlyRole
proposeMint	External	Can Modify State	onlyRole
executeMint	External	Can Modify State	onlyRole
crossoutBurn	External	Payable	-
getPeriodInfo	External	-	-

4.3 Vulnerability Summary

[N1] [High] The validity of the token contract address is not checked

Category: Design Logic Audit

Content

In the BitlayerBridgeV2 contract, the `crossoutBurn` function does not check the validity of the `token` address. An attacker can pass in the token contract address created by himself, and then transfer and burn the tokens created by the attacker through the crossoutBurn function to trigger the `CrossoutBurned` event.

contracts/BitlayerBridgeV2.sol#L278-L307

```
function crossoutBurn(
    address token,
```

```
        uint256 value,
        string memory btcReceiver
    )
        external
        payable
    {
        require(token != address(0), "invalid token address");
        require(value != 0, "invalid value");
        require(bytes(btcReceiver).length != 0, "invalid btcReceiver");
        uint256 crossoutFeeAmount = tokenConfigs[token].crossoutFee;
        require(crossoutFeeAmount == 0 || msg.value >= crossoutFeeAmount, "not enough
fee paied");

        SafeERC20.safeTransferFrom(IERC20(token), msg.sender, address(this), value);
        IPegToken(token).burn(address(this), value);

        if (crossoutFeeAmount > 0) {
            (bool success, bytes memory returndata) = feeAddress.call{value:
crossoutFeeAmount}("");
            require(success, string(returndata));
        }

        uint256 unusedFee = msg.value - crossoutFeeAmount;
        if (unusedFee > 0) {
            (bool success, bytes memory returndata) = msg.sender.call{value:
unusedFee}("");
            require(success, string(returndata));
        }

        emit CrossoutBurned(msg.sender, btcReceiver, token, value);
    }
```

Solution

It is recommended to add a token whitelist to the contract and check the incoming token contract in the crossoutBurn function to check whether the token contract is in the whitelist.

Status

Fixed

[N2] [Medium] Unlimited huge amount minting

Category: Design Logic Audit

Content

In the BitlayerBridgeV2 contract, when huge amounts are minted through the `proposeMint` function and

`executeMint` function, the minted amount is not limited by `periodLimit`, nor is it recorded in `periodMinted`, and the `period` will not be updated.

contracts/BitlayerBridgeV2.sol#L235-L255,L258-L275

```
    function proposeMint(
        MintInfo memory mInfo
    )
        external
        onlyRole(ProposerRole)
    {
        bytes32 blExecHash = keccak256(abi.encodePacked(mInfo.btcTxHash));
        require(!executedBtcHash[blExecHash], "already executed");
        require(mInfo.receiver != address(0), "receiver is null");
        require(proposedMint[blExecHash].receiver == address(0), "already proposed");
        require(mInfo.value >= tokenConfigs[mInfo.token].mintSplitLine, "mint value is
less than configed");

        require(
            !IPegToken(mInfo.token).isBlacklist(mInfo.receiver),
            "receiver is blacklisted"
        );

        proposedMint[blExecHash] = mInfo;

        emit MintProposed(mInfo.btcTxHash, blExecHash, mInfo.receiver, mInfo.token,
mInfo.value);
    }

    function executeMint(
        bytes32 blExecHash
    )
        external
        onlyRole(ExecutorRole)
    {
        MintInfo memory mInfo = proposedMint[blExecHash];

        require(!executedBtcHash[blExecHash], "already executed");
        require(mInfo.receiver != address(0), "not proposed");

        executedBtcHash[blExecHash] = true;
        delete proposedMint[blExecHash];

        IPegToken(mInfo.token).mint(mInfo.receiver, mInfo.value);

        emit MintExecuted(mInfo.btcTxHash, blExecHash, mInfo.receiver, mInfo.token,
```

```
mInfo.value);
    }
```

Solution

It is recommended to check whether this design meets the design expectations and add restrictions on huge minting if it does not.

Status

Acknowledged; The project team said the design met expectations.

[N3] [Low] The returned leftQuota value is inaccurate

Category: Design Logic Audit

Content

In the BitlayerBridgeV2 contract, the `getPeriodInfo` function is inaccurate when returning the `leftQuota` of the specified token. If `block.number` has reached the next period and `tokenConfigs[token]` has not been updated, the value of `leftQuota` should be `config.periodLimit`.

contracts/BitlayerBridgeV2.sol#L309-L319

```
    function getPeriodInfo(address token)
        external
        view
        returns(uint256 periodLimit, uint256 leftQuota, uint256
leftBlocksToNextPeriod)
    {
        TokenConfig memory config = tokenConfigs[token];

        periodLimit = config.periodLimit;
        leftQuota = config.periodLimit - config.periodMinted;
        leftBlocksToNextPeriod = config.periodInterval - (block.number -
config.lastBlockNumber) % config.periodInterval;
    }
```

Solution

It is recommended to determine whether block.number is greater than lastBlockNumber + periodInterval. If it is greater, then leftQuota = config.periodLimit. If it is less, return leftQuota = config.periodLimit - config.periodMinted.

Status

Fixed

[N4] [Suggestion] Missing zero address check

Category: Others

Content

1.In the BitlayerBridgeV2 contract, the `initToken` function does not perform a zero address check on the `token` address.

contracts/BitlayerBridgeV2.sol#L99-L119

```
function initToken(
    address token,
    ```
)
    external
    onlyRole(AdminRole)
{
    TokenConfig storage config = tokenConfigs[token];
    ```
}
```

2.In the BitlayerBridgeV2 contract, the `directMint` function does not perform a zero address check on the `mInfo.token` address.

contracts/BitlayerBridgeV2.sol#L198-L233

```
function directMint(
    MintInfo memory mInfo
)
    external
    onlyRole(MinterRole)
{
    ```
    IPegToken(mInfo.token).mint(mInfo.receiver, mInfo.value);
    ```
}
```

3.In the BitlayerBridgeV2 contract, the `proposeMint` function does not perform a zero address check on the `mInfo.token` address.

BitlayerBridgeV2.sol#L235-L255

```
function proposeMint(
    MintInfo memory mInfo
)
    external
    onlyRole(ProposerRole)
{
    ```
    require(
        !IPegToken(mInfo.token).isBlacklist(mInfo.receiver),
        "receiver is blacklisted"
    );
    ```
}
```

Solution

It is recommended to add zero address check in the function.

Status

Fixed

[N5] [Suggestion] Unchecked parameter value is not 0

Category: Others

Content

1.In the BitlayerBridgeV2 contract, the `initToken` function does not check whether the values of `periodInterval` and `mintSplitLine` are not 0.

contracts/BitlayerBridgeV2.sol#L99-L119

```
function initToken(
    address token,
    uint256 periodLimit,
    uint256 periodInterval,
    uint256 mintSplitLine,
    uint256 crossoutFee
)
```

```
        external
        onlyRole(AdminRole)
    {
        TokenConfig storage config = tokenConfigs[token];
        require(config.lastBlockNumber == 0, "already inited");

        config.lastBlockNumber = block.number;
        config.periodLimit     = periodLimit;
        config.periodInterval  = periodInterval;
        config.mintSplitLine   = mintSplitLine;
        config.crossoutFee     = crossoutFee;

        emit TokenInited(token, periodLimit, periodInterval, mintSplitLine,
crossoutFee);
    }
```

2.In the BitlayerBridgeV2 contract, the `setMintSplitLine` function does not check whether the value of `limit` is not 0.

contracts/BitlayerBridgeV2.sol#L137-L149

```
    function setMintSplitLine(
        address token,
        uint256 limit
    )
        public
        onlyRole(AdminRole)
    {
        require(token != address(0), "invalid token address");

        tokenConfigs[token].mintSplitLine = limit;

        emit MintSplitLineSet(token, limit);
    }
```

3.In the BitlayerBridgeV2 contract, the `directMint` function does not check whether the value of `mInfo.value` is not 0.

BitlayerBridgeV2.sol#L198-L233

```
    function directMint(
        MintInfo memory mInfo
    )
        external
```

```
    onlyRole(MinterRole)
{
    ```
    require(
        mInfo.value < config.mintSplitLine,
        "mint value is greater than configed"
    );
    ```
}
```

Solution

It is recommended to check whether the parameter value is not 0 in the function.

Status

Fixed

[N6] [Suggestion] Redundant code

Category: Others

Content

In the BitlayerBridgeV2 contract, the `directMint` function repeatedly performs a zero address check on the `mInfo.receiver` address because the zero address check has already been performed on this address in the `mint` function of the token contract.

contracts/BitlayerBridgeV2.sol#L198-L233

```
function directMint(
    MintInfo memory mInfo
)
    external
    onlyRole(MinterRole)
{
    ```
    require(mInfo.receiver != address(0), "receiver is null");
    ```
}
```

Solution

It is recommended to remove redundant code.

Status

Acknowledged; The project team stated that the zero address check of the mint function was too deep, so this check was added to the directMint function.

[N7] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the BitlayerBridgeV2 contract, the `AdminRole` role can set important parameters in the contract through the following functions.

contracts/BitlayerBridgeV2.sol

```
function setRoles
function initToken
function setPeriodMintLimit
function setMintSplitLine
function setCrossoutFee
function setPeriodInterval
function startNewPeriod
```

2.The BitlayerBridge contracts are implemented using the OpenZeppelin upgradeable model, allowing the `AdminRole` role to perform contract upgrades.Since the BitlayerBridgeV2 contract inherits the BitlayerBridge contract, it is also an upgradeable contract. However, this design introduces an excessive privilege risk.

contracts/BitlayerBridge.sol#L4

```
import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
```

3.In the BitlayerBridge contract, `AdminRole` role can set important parameters of the contract.And the BitlayerBridgeV2 contract also inherits these privileged functions.

contracts/BitlayerBridge.sol

```
function setFeeAddress
function setLockFee
function setMinLockAmount
function setMaxLockAmount
```

4.In the BitlayerBridge contract, the `LiquidityRole` role can call the `removeLiquidityTo` function to remove the liquidity of any address.And the BitlayerBridgeV2 contract also inherits this privileged functions.

contracts/BitlayerBridge.sol#L159-L162

```
function removeLiquidity(uint256 amount) external whenNotPaused {
    require(amount > 0, "invalid amount");
    doRemoveLiquidity(msg.sender, amount);
}
```

5.In the BitlayerBridge contract, `UnlockRole` can pass in any `_txHash` through the `unlock` function to unlock any amount of native tokens. In the BitlayerBridgeV2 contract, `MinterRole`, `ProposerRole` and `ExecutorRole` can mint any amount of PegToken by passing in any `btcTxHash` parameter through their respective functions (`directMint`, `proposeMint` and `executeMint`). These vulnerabilities stem from the contract's inability to verify Bitcoin network's transaction hash at the EVM level, instead leaving verification to a centralized validator.

contracts/BitlayerBridge.sol

```
function unlock
```

contracts/BitlayerBridgeV2.sol

```
function directMint
function proposeMint
function executeMint
```

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. 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

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002405140001	SlowMist Security Team	2024.05.13 - 2024.05.14	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 high risk, 2 medium risk, 1 low risk, 3 suggestion.

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