Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.03.29, the SlowMist security team received the Bitlayer team's security audit application for Bitlayer Bridge, 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 project is BTC Layer2 Bridge contract.

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 Audit	Medium	Acknowledged
N2	Missing zero address check	Others	Suggestion	Fixed

NO	Title	Category	Level	Status
N3	Return value not checked	Others	Suggestion	Acknowledged
N4	Parameter Validation Missing in unlock Function	Design Logic Audit	Medium	Acknowledged
N5	Min lock amount not checked against max	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

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

Initial audit commit: 1c4f70e54faebf8a27e17a9fac8ffa83982c0205

(Focus on code changes from version: 41c7c064117218eef147f4ae7e7052708846273d to version:

1c4f70e54faebf8a27e17a9fac8ffa83982c0205)

Final audit commit: d8375e8baa8ac6be93edff7d76c11c8b991a26c7

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:

BitlayerBridge			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	Public	Can Modify State	initializer
_authorizeUpgrade	Internal	Can Modify State	onlyRole

BitlayerBridge			
pause	External	Can Modify State	onlyRole whenNotPaused
unpause	External	Can Modify State	onlyRole whenPaused
setFeeAddress	External	Can Modify State	onlyRole
setLockFee	External	Can Modify State	onlyRole
setMinLockAmount	External	Can Modify State	onlyRole
setMaxLockAmount	External	Can Modify State	onlyRole
<Receive Ether>	External	Payable	-
doRemoveLiquidity	Internal	Can Modify State	-
removeLiquidity	External	Can Modify State	whenNotPaused
removeLiquidityTo	External	Can Modify State	onlyRole whenNotPaused
lock	External	Payable	whenNotPaused
unlock	External	Can Modify State	onlyRole whenNotPaused

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.The BitlayerBridge contracts are implemented using the OpenZeppelin upgradeable model, allowing the `AdminRole` role to perform contract upgrades. However, this design introduces an excessive privilege risk.

2.In the BitlayerBridge contract, the `UnlockRole` role can call the `unlock` function to unlock the ETH locked in the contract; the `LiquidityRole` role can call the `removeLiquidityTo` function to remove the liquidity of any address.

BitlayerBridge.sol#L125-L134,L148-L163

```
    function removeLiquidityTo(address to, uint256 amount)
        external
        onlyRole(LiquidityRole)
        whenNotPaused
    {
        ```
    }

    function unlock(string memory _txHash, address to, uint256 amount)
        external
        onlyRole(UnlockRole)
        whenNotPaused
    {
        ```
    }
}
```

3.In the BitlayerBridge contract, `AdminRole` role can set important parameters of the contract.

BitlayerBridge.sol

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

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

[N2] [Suggestion] Missing zero address check

Category: Others

Content

In the BitlayerBridge contract, the zero address check is missing in the `initialize` function and `unlock` function.

BitlayerBridge.sol#L36-L75,L148-L163

```
    function initialize(
    ```
)
        public
        initializer
    {
    ```
    feeAddress = _feeAddress;
    ```
    }

    function unlock(string memory _txHash, address to, uint256 amount)
        external
        onlyRole(UnlockRole)
        whenNotPaused
    {
        ```
        (bool success, bytes memory returndata) = payable(to).call{value: amount}
("");
        ```
    }
```

**Solution**

It's recommended to add the check whether the account is a zero address.

**Status**

Fixed; The project side stated that the to parameter of the unlock function is specified in an OP_RETURN UTXO of the transaction on the Bitcoin chain.

### [N3] [Suggestion] Return value not checked

**Category: Others**

**Content**

In the BitlayerBridge contract, the return value is not checked when the `initialize` function calls the `_grantRole` function.

BitlayerBridge.sol#L36-L75
```

```
    function initialize(
        ```
)
        public
        initializer
    {
        _grantRole(AdminRole, admin);
        ```
    }
```

Solution

It is recommended to check the return value.

Status

Acknowledged

[N4] [Medium] Parameter Validation Missing in unlock Function

Category: Design Logic Audit

Content

In the BitlayerBridge contract, the `unlock` function does not verify the validity of the parameters passed in. The `UnlockRole` role can enter any `_txHash` (not recorded by `txUnlocked` mapping) and `amount` to unlock the ETH in the contract and transfer it to the specified address.

BitlayerBridge.sol#L148-L163

```
    function unlock(string memory _txHash, address to, uint256 amount)
        external
        onlyRole(UnlockRole)
        whenNotPaused
    {
        bytes32 txHash = keccak256(abi.encode(_txHash));
        require(!txUnlocked[txHash], "txHash already unlocked");
        txUnlocked[txHash] = true;

        (bool success, bytes memory returndata) = payable(to).call{value: amount}
("");
        require(success, string(returndata));

        totalUnlocked += amount;
```

```
        emit NativeUnlocked(_txHash, to, amount);
    }
```

Solution

It is recommended to verify the _txHash, to, and amount parameters to ensure that _txHash is valid and corresponds to the to address and amount parameters.

Status

Acknowledged; The project team stated that the unlock function is called by the relay operator of the cross-chain bridge. The address used by the relay operator is a multi-signature addressusing MPC (Multi-Party Computation). Additionally, there will be an independent risk control system to verify and ensure the secure and correct transmission and execution of cross-chain messages.

[N5] [Low] Min lock amount not checked against max

Category: Design Logic Audit

Content

In the BitlayerBridge contract, whether the variable `minLockAmount` is less than the variable `maxLockAmount` is not checked in the `initialize` function, `setMinLockAmount` function, and `setMaxLockAmount` function. If the variable `minLockAmount` is greater than the variable `maxLockAmount` , the `lock` function cannot be used.

BitlayerBridge.sol#L41-L85,L113-L118,L120-L125

```
    function initialize(
        ```
)
        public
        initializer
    {
        ```
        minLockAmount = _minLockAmount;
        maxLockAmount = _maxLockAmount;
    }

    function setMinLockAmount(uint256 min) external onlyRole(AdminRole) {
        uint256 oldMin = minLockAmount;
        minLockAmount = min;

        emit MinLockAmountSet(oldMin, min);
    }
```

```
function setMaxLockAmount(uint256 max) external onlyRole(AdminRole) {
    uint256 oldMax = maxLockAmount;
    maxLockAmount = max;

    emit MaxLockAmountSet(oldMax, max);
}
```

Solution

It is recommended to check whether the variable minLockAmount is less than the variable maxLockAmount in the initialize function, setMinLockAmount function, and setMaxLockAmount function.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404020001	SlowMist Security Team	2024.03.29 - 2024.04.02	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 medium risk, 1 low risk, 2 suggestion. All findings have been fixed or acknowledged. 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist