Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.12.05, the SlowMist security team received the BitFi team's security audit application for BitFi - BFBTC, 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 BitFi's BFBTC contract, which mainly includes proxy and vault modules.

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 event records	Others	Suggestion	Acknowledged

NO	Title	Category	Level	Status
N3	Redundant code	Others	Suggestion	Fixed
N4	Missing zero address check	Others	Suggestion	Acknowledged
N5	The whenNotPaused modifier is not added	Others	Suggestion	Fixed
N6	Deposit amplification through msg.value reuse in multicall	Design Logic Audit	Critical	Fixed
N7	Missing zero address check	Others	Suggestion	Fixed
N8	The calculation logic only uses BTC pegged tokens	Others	Information	Acknowledged
N9	BTC transaction hash cannot be verified for validity	Others	Information	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/bitfi-labs/contract

Initial audit version: 47d330ad5f01c306af98a037dfc8038b6b7c6d9d

Final audit version: e79c83d2afb702bbfeaa39c05b8e7fde11f0cc1b

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:

BfbtcBitlayer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setUnderlyingWallet	External	Can Modify State	onlyOwner
setBridge	External	Can Modify State	onlyOwner
setMultisig	External	Can Modify State	onlyOwner
_checkMultisig	Internal	-	-
decimals	Public	-	-
setPause	External	Can Modify State	onlyOwner
multicall	Public	Payable	-
currentRatio	Public	-	-
previewDeposit	Public	-	-
deposit	External	Payable	whenNotPaused
requestWithdraw	External	Can Modify State	-
approveWithdraw	External	Can Modify State	whenNotPaused onlyMultisig
approveUnbound	External	Can Modify State	whenNotPaused
previewWithdraw	Public	-	-
updateEpoch	External	Can Modify State	whenNotPaused onlyMultisig
_claimable	Internal	-	-
claimable	External	-	-
claim	External	Can Modify State	whenNotPaused
safeTransferETH	Internal	Can Modify State	-

BfbtcBitlayer			
setCooldownEpoches	External	Can Modify State	onlyOwner
_authorizeUpgrade	Internal	Can Modify State	onlyOwner

Bfbtc			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
setMultisig	External	Can Modify State	onlyOwner
setUnderlyingWallet	External	Can Modify State	onlyOwner
_checkMultisig	Internal	-	-
decimals	Public	-	-
setPause	External	Can Modify State	onlyOwner
multicall	Public	Payable	-
getLatestPrice	Public	-	-
currentRatio	Public	-	-
previewDeposit	Public	-	-
deposit	External	Can Modify State	whenNotPaused
depositNative	External	Can Modify State	whenNotPaused onlyMultisig
requestWithdraw	External	Can Modify State	-
requestWithdrawNative	External	Can Modify State	whenNotPaused
previewWithdraw	Public	-	-
approveWithdraw	External	Can Modify State	whenNotPaused onlyMultisig
nativeWithdraw	External	Can Modify State	whenNotPaused onlyMultisig

Bfbtc			
updateEpoch	External	Can Modify State	whenNotPaused onlyMultisig
_claimable	Internal	-	-
claimable	External	-	-
claim	External	Can Modify State	whenNotPaused
setCooldownEpoches	External	Can Modify State	onlyOwner
setStalePriceDelay	External	Can Modify State	onlyOwner
_authorizeUpgrade	Internal	Can Modify State	onlyOwner

BfbtcProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC1967Proxy

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the BfbtcBitlayer contract, the `Owner` role can modify important parameters in the function through the following functions.

- contracts/BfbtcBitlayer.sol#L102-L104, L106-L108, L110-L112, L127-L134, L265-L267

```
function setUnderlyingWallet
function setBridge
function setMultisig
function setPause
function setCooldownEpoches
```

2.In the BfbtcBitlayer contract, the `Multisig` role can approve the user's withdrawal request through the `approveWithdraw` function, and modify the `currentEpoch` parameter and the corresponding `ratio` through the `updateEpoch` function.

- contract/contracts/BfbtcBitlayer.sol#L174-L195, L215-L220

```
    function approveWithdraw
    function updateEpoch
```

3.In the BfbtcBitlayer contract, the `bridge` role can approve the unbinding request through the `approveUnbound` function.

- contracts/BfbtcBitlayer.sol#L196-L209

```
    function approveUnbound
```

4.Both BfbtcBitlayer and Bfbtc contracts inherit OpenZeppelin's UUPSUpgradeable pattern, where upgrade authority is concentrated in the Owner role, creating potential security vulnerabilities due to privileged access control.

5.In the Bfbtc contract, the `Owner` role can modify important parameters in the function through the following functions.

- contracts/Bfbtc.sol#L113-L115, L117-L119, L134-L141, L332-L334, L336-L338

```
    function setMultisig
    function setUnderlyingWallet
    function setPause
    function setCooldownEpoches
    function setStalePriceDelay
```

6.In the Bfbtc contract, the `Multisig` role can perform deposit and withdrawal operations through the `depositNative` function, `approveWithdraw` function, and `nativeWithdraw` function; and modify the `currentEpoch` parameter, `ratio`, and `underlyingPrice` mapping through the updateEpoch function.

- contracts/Bfbtc.sol#L191-L208, L253-L272, L274-L299, L301-L308

```
function depositNative
function approveWithdraw
function nativeWithdraw
function updateEpoch
```

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

[N2] [Suggestion] Missing event records

Category: Others

Content

In the Bfbtc contract and BfbtcBitlayer contract, the following functions modify important parameters in the contract, but there is a lack of event records.

- contracts/BfbtcBitlayer.sol#L102-L104, L106-L108, L110-L112, L265-L267

```
function setUnderlyingWallet
function setBridge
function setMultisig
function setCooldownEpoches
```

- contracts/Bfbtc.sol#L113-L115, L117-L119, L332-L334, L336-L338

```
function setMultisig
function setUnderlyingWallet
function setCooldownEpoches
function setStalePriceDelay
```

Solution

It is recommended to add event logging.

Status

Acknowledged

[N3] [Suggestion] Redundant code

Category: Others

Content

In the BfbtcBitlayer contract, the following parameters and errors are not used.

- contract/contracts/BfbtcBitlayer.sol#L33, L69, L70, L72, L73, L76

```
    address public underlyingWallet;
    error ZeroMintAmount();
    error InvalidBTCAddressType();
    error InvalidOraclePrice();
    error StalePriceData();
    error InvalidData();
```

Solution

It is recommended to delete the redundant code.

Status

Fixed

[N4] [Suggestion] Missing zero address check

Category: Others

Content

In the Bfbtc contract and BfbtcBitlayer contract, the `initialize` function lacks a zero address check for the address parameter.

- contracts/Bfbtc.sol#L90-L111

```
    function initialize
```

- contracts/BfbtcBitlayer.sol#L87-L100

```
    function initialize
```

Solution

It is recommended to add a zero address check.

Status

Acknowledged

[N5] [Suggestion] The whenNotPaused modifier is not added

Category: Others

Content

The `whenNotPaused` modifier is not added to the `requestWithdraw` function in the BfbtcBitlayer contract and the Bfbtc contract.

- contracts/Bfbtc.sol#L210-L226

```
    function requestWithdraw(uint256 amount) external {
        //...
    }
```

- contracts/BfbtcBitlayer.sol#L175-L188

```
    function requestWithdraw(uint256 amount) external {
        //...
    }
```

Solution

It is recommended to add the whenNotPaused modifier.

Status

Fixed

[N6] [Critical] Deposit amplification through msg.value reuse in multicall

Category: Design Logic Audit

Content

The `multicall` function in the Bfbtc contract allows calling multiple contract functions in the same transaction. When a user calls the `deposit` function multiple times through `multicall`, the `msg.value` parameter will be reused in each `delegatecall`, which may cause the user to complete multiple `deposit` operations when the user actually only paid the native token once. For example, if an attacker sends 1 native token and calls `deposit` 3 times through `multicall`, the attacker may eventually mint 3 times the amount of tokens, which will cause serious loss of protocol funds.

- contracts/BfbtcBitlayer.sol#L136-L150, L160-L173

```
    function multicall(bytes[] calldata data) public payable returns (bytes[] memory
results) {
        results = new bytes[](data.length);
        for (uint256 i; i < data.length; ++i) {
            (bool success, bytes memory result) = address(this).delegatecall(data[i]);

            if (!success) {
                if (result.length == 0) revert();
                assembly {
                    revert(add(0x20, result), mload(result))
                }
            }

            results[i] = result;
        }
    }

    function deposit(uint256 minAmount) external payable whenNotPaused {
        uint256 amount = msg.value;
        if (amount == 0) revert ZeroDepositAmount();

        uint256 mintAmount = previewDeposit(amount);
        if (mintAmount < minAmount) revert MinimumAmountNotMet();

        _mint(msg.sender, mintAmount);

        uint256 epoch = currentEpoch;
        emit Deposit(msg.sender, epoch, amount, mintAmount);
        bytes memory data;
```

```
        emit TokenStaked(++reqId, msg.sender, amount, address(0), epoch, 0, data);
    }
```

Solution

It is recommended to globally accumulate the value in the deposit function by recording the initial msg.value value before multicall execution and deducting the used amount from it after each deposit operation to ensure that the sum of msg.value for all deposit operations does not exceed the amount of navive token actually paid by the user.

Status

Fixed

[N7] [Suggestion] Missing zero address check

Category: Others

Content

In the Bfbtc contract and BfbtcBitlayer contract, the following function lacks a zero address check for the address parameter.

- contracts/Bfbtc.sol#L113-L115, L117-L119, L191-L208

```
    function setMultisig
    function setUnderlyingWallet
    function depositNative
```

- contracts/BfbtcBitlayer.sol#L102-L104, L106-L108, L110-L112

```
    function setUnderlyingWallet
    function setBridge
    function setMultisig
```

Solution

It is recommended to add a zero address check.

Status

Fixed

[N8] [Information] The calculation logic only uses BTC pegged tokens

Category: Others

Content

In the Bfbtc contract, the `previewDeposit` function is used to calculate the amount of bfBTC that can be minted, but its calculation formula only applies to ERC20 tokens pegged to BTC.

- contracts/Bfbtc.sol#L175-L177

```
    function previewDeposit(uint256 amount) public view returns (uint256) {
        return (amount * getLatestPrice(true) * currentRatio()) / PRICE_PRECISION /
(10 ** underlyingDecimals);
    }
```

Solution

It is recommended that when the underlying asset is not a token pegged to BTC, the calculation logic needs to be redesigned.

Status

Acknowledged

[N9] [Information] BTC transaction hash cannot be verified for validity

Category: Others

Content

Because the validity of the transaction hash of the BTC network cannot be verified in the smart contract, the validity of the `nativeTxs` parameter cannot be verified in the `depositNative` function and `nativeWithdraw` function in the Bfbtc contract.

- contracts/Bfbtc.sol#L191-L208, L274-L299

```
    function depositNative(
        address[] calldata users,
        uint256[] calldata amounts,
        bytes32[] calldata nativeTxs
    ) external whenNotPaused onlyMultisig {
        //...
        for (uint256 i; i < users.length; ++i) {
            if (amounts[i] == 0) revert ZeroMintAmount();
            if (usedNativeTx[nativeTxs[i]]) revert NativeTxAlreadyUsed();
```

```
            usedNativeTx[nativeTxs[i]] = true;
            uint256 mintAmount = (amounts[i] * cachedRatio) / (10 ** 8);
            _mint(users[i], mintAmount);

            emit DepositNative(users[i], currentEpoch, amounts[i], mintAmount,
nativeTxs[i]);
        }
    }

    function nativeWithdraw(
        uint256 epoch,
        uint256[] calldata withdrawIds,
        bytes32[] calldata nativeTxs
    ) external whenNotPaused onlyMultisig {
        //...
        for (uint256 i; i < withdrawIds.length; ++i) {
            Withdrawal storage withdrawal = withdrawals[withdrawIds[i]];
            if (withdrawal.status != WITHDRAWAL_STATUS_PENDING) revert
InvalidWithdrawalStatus();
            if (epoch < withdrawal.epoch + cooldownEpoches) revert
CooldownPeriodNotPassed();

            if (withdrawal.btcAddressType == 0) {
                revert InvalidBTCAddressType();
            } else {
                withdrawal.status = WITHDRAWAL_STATUS_SENT_ON_NATIVE;
                withdrawal.settleEpoch = epoch;
                withdrawal.nativeTx = nativeTxs[i];
                totalShares += withdrawal.amount;
            }

            emit WithdrawNative(withdrawIds[i], epoch, nativeTxs[i]);
        }
        //...
    }
```

Solution

It is recommended to reduce the risk of double spending by deploying a multi-signature system and sufficient number of confirmations.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002412060005	SlowMist Security Team	2024.12.05 - 2024.12.06	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 critical risk, 1 medium risk, 5 suggestion, 2 information.

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