Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.06.27, the SlowMist security team received the Cyber team's security audit application for Cyber Token Bridges Phase2, 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 audit only includes the TokenAdapter and TokenController modules of the Cyber Token Bridges protocol.The contracts primarily inherit from the OFTAdapter and OFTCore contracts in the third-party library provided by layerzerolabs.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Token compatibility	Design Logic	Low	Acknowledged

NO	Title	Category	Level	Status
	issues	Audit		
N2	Missing zero address validation	Others	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

CyberTokenAdapter ETH: 0xCB07992DE144bDeE56fDb66Fff2454B43243b052

CyberTokenController Cyber: 0x9a9d5a29206dde4f70825032df32333de5f63921

CyberTokenController BSC: 0x9a9d5a29206dde4f70825032df32333de5f63921

CyberTokenController OP Mainnet: 0x9a9d5a29206dde4f70825032df32333de5f63921

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

CyberTokenAdapter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OFTAdapter Ownable

CyberTokenController			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OFTCore Ownable
token	Public	-	-
approvalRequired	External	-	-
_debit	Internal	Can Modify State	-

CyberTokenController			
_credit	Internal	Can Modify State	-
transferTokenOwnership	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Low] Token compatibility issues

Category: Design Logic Audit

Content

When the _debit function is called, it calculates the amount of tokens to be burned and the estimated amount of tokens to be received based on the _amountLD parameter passed by the user. These two values are returned at the end of the function call. However, the estimated amount of tokens to be received is not calculated as the difference between the balance before and after the user burns the tokens; instead, it is directly equal to the amount of tokens to be burned. This could lead to a discrepancy between the actual amount of tokens received and the calculated estimate, especially if the token being burned is deflationary or inflationary, potentially causing errors in cross-chain accounting. The _credit function has the same issue, where the estimated amount of tokens to be received is directly computed as the _amountLD parameter passed in.

Code Location:

src/CyberTokenController.sol#L67-107

```
    function _debit(
        address _from,
        uint256 _amountLD,
        uint256 _minAmountLD,
        uint32 _dstEid
    )
        internal
        virtual
        override
        returns (uint256 amountSentLD, uint256 amountReceivedLD)
    {
        (amountSentLD, amountReceivedLD) = _debitView(
            _amountLD,
            _minAmountLD,
```

```
            _dstEid
        );
        // @dev Burn tokens
        IMintableBurnable(innerToken).burnFrom(_from, amountSentLD);
    }

    ...
    function _credit(
        address _to,
        uint256 _amountLD,
        uint32 /*_srcEid*/
    ) internal virtual override returns (uint256 amountReceivedLD) {
        // @dev Mint the tokens to the recipient.
        IMintableBurnable(innerToken).mint(_to, _amountLD);
        // @dev In the case of NON-default CyberTokenController, the amountLD MIGHT
not be == amountReceivedLD.
        return _amountLD;
    }
```

Solution

It is recommended to calculate the amount of tokens received across chains based on the difference in the user's balance before and after token burning or minting. This approach helps prevent distortions caused by deflationary or inflationary tokens.

Status

Acknowledged

[N2] [Suggestion] Missing zero address validation

Category: Others

Content

In the CyberTokenController contract, the constructor initialization operation does not check whether the _token and _lzEndpoint address are zero. Furthermore, when the Owner role calls the transferTokenOwnership function to transfer the management rights of the innerToken, there is no check to ensure that the new owner address is not a zero address.

Code location:

src/CyberTokenController.sol#L21-30

```
    constructor(
        address _token,
        address _lzEndpoint,
        address _delegate
    )
        OFTCore(IERC20Metadata(_token).decimals(), _lzEndpoint, _delegate)
        Ownable(_delegate)
    {
        innerToken = _token;
    }
```

src/CyberTokenController.sol#L109-111

```
    function transferTokenOwnership(address newOwner) external onlyOwner {
        Ownable(innerToken).transferOwnership(newOwner);
    }
```

Solution

It is recommended to add the zero address check.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002406270001	SlowMist Security Team	2024.06.27 - 2024.06.27	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 low risk and 1 suggestion vulnerabilities. All the findings were Acknowledged. The code has been 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