Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2024.06.11, the SlowMist security team received the Cyber team's security audit application for Cyber Token Bridges, 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 focuses on the CyberVault and CyberStakingPool modules of Cyber's Cyber Token Bridges protocol. Users can stake Cyber tokens through the CyberStakingPool contract and earn rewards. CyberVault inherits the ERC4626 protocol and is used to manage the Cyber tokens deposited by users and centrally stake them into the CyberStakingPool contract.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing event records	Others	Suggestion	Acknowledged
N2	Optimizable setLockDuration check	Design Logic Audit	Suggestion	Acknowledged
N3	CyberVault withdrawals have flaws	Design Logic Audit	Critical	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/cyberconnecthq/cyber-token-bridges

commit: e4f8823d046289e018ee7fc6b446f95aa1b8d32c

Audit Scope:

- src/CyberStakingPool.sol
- src/CyberVault.sol

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:

CyberStakingPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
_update	Internal	Can Modify State	-

CyberStakingPool			
_collectFee	Internal	Can Modify State	-
_authorizeUpgrade	Internal	-	-
minimalStakeAmount	External	-	-
stake	External	Can Modify State	updateReward
unstake	External	Can Modify State	updateReward
withdraw	External	Can Modify State	updateReward
claimAllRewards	External	Can Modify State	updateReward
rewardBalance	External	-	-
claimableAllRewards	External	-	-
getLockedAmountByKey	External	-	-
lockedAmountByUser	External	-	-
circulatingSupply	Public	-	-
rewardBalanceKey	Public	-	-
claimableRewards	Public	-	-
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
setLockDuration	External	Can Modify State	onlyOwner
setMinimalStakeAmount	External	Can Modify State	onlyOwner
setProtocolFeeBps	External	Can Modify State	onlyOwner
claimProtocolFee	External	Can Modify State	onlyOwner
_claimReward	Private	Can Modify State	-
_updateCurrentUnclaimedRewards	Private	Can Modify State	-

CyberVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
totalAssets	Public	-	-
previewDeposit	Public	-	-
previewMint	Public	-	-
previewWithdraw	Public	-	-
previewRedeem	Public	-	-
deposit	Public	Can Modify State	-
mint	Public	Can Modify State	-
withdraw	Public	Can Modify State	-
redeem	Public	Can Modify State	-
_withdraw	Internal	Can Modify State	-
_update	Internal	Can Modify State	-
decimals	Public	-	-
_authorizeUpgrade	Internal	-	-
initiateRedeem	External	Can Modify State	-
initiateWithdraw	External	Can Modify State	-
getLockAmount	External	-	-
stake	Public	Can Modify State	-
claim	Public	Can Modify State	-
claimAndStake	Public	Can Modify State	-

CyberVault			
batchDeposit	External	Can Modify State	-
setLockDuration	External	Can Modify State	onlyOwner
setProtocolFeeBps	External	Can Modify State	onlyOwner
setProtocolFeeTreasury	External	Can Modify State	onlyOwner
_initiateWithdraw	Private	Can Modify State	-
_convertToShares	Private	-	-
_convertToAssets	Private	-	-
_totalAssetsWithoutFee	Private	-	-
_previewDepositInternal	Private	-	-
_previewMintInternal	Private	-	-
_previewWithdrawInternal	Private	-	-
_previewRedeemInternal	Private	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event records

Category: Others

Content

In the CyberStakingPool contract, the owner role can modify the lockDuration, _minimalStakeAmount, and protocolFeeBps parameters through the setLockDuration, setMinimalStakeAmount, and setProtocolFeeBps functions, respectively. However, no event logging is performed.

Similarly, in the CyberVault contract, the owner role can modify the lockDuration, protocolFeeBps, and protocolFeeTreasury parameters through the setLockDuration, setProtocolFeeBps, and setProtocolFeeTreasury functions, respectively. However, no event logging is performed.

Code location:

src/CyberStakingPool.sol#L330-L343

```
    function setLockDuration(uint256 _lockDuration) external onlyOwner {
        lockDuration = _lockDuration;
    }

    function setMinimalStakeAmount(
        uint256 minimalStakeAmount_
    ) external onlyOwner {
        _minimalStakeAmount = minimalStakeAmount_;
    }

    function setProtocolFeeBps(uint256 _protocolFeeBps) external onlyOwner {
        require(_protocolFeeBps <= MAX_BPS, "INVALID_PROTOCOL_FEE_BPS");
        protocolFeeBps = _protocolFeeBps;
    }
```

src/CyberVault.sol#L330-L343

```
    function setLockDuration(uint256 _lockDuration) external onlyOwner {
        lockDuration = _lockDuration;
    }

    function setProtocolFeeBps(uint256 _protocolFeeBps) external onlyOwner {
        require(_protocolFeeBps <= MAX_BPS, "INVALID_PROTOCOL_FEE_BPS");
        protocolFeeBps = _protocolFeeBps;
    }

    function setProtocolFeeTreasury(
        address _protocolFeeTreasury
    ) external onlyOwner {
        protocolFeeTreasury = _protocolFeeTreasury;
    }
```

Solution

It is recommended to implement event logging when modifying sensitive parameters to facilitate subsequent self-inspection or community auditing.

Status

Acknowledged

[N2] [Suggestion] Optimizable setLockDuration check

Category: Design Logic Audit

Content

In the CyberVault contract, the owner can modify the lockDuration parameter through the setLockDuration function. Theoretically, the value of the lockDuration parameter should be the same as the lockDuration value in CyberStakingPool, so that users of CyberVault can withdraw funds after the withdrawal lock is completed in CyberStakingPool. However, the setLockDuration function in the CyberVault contract does not check whether the set lockDuration value is the same as the lockDuration value in CyberStakingPool. If an incorrect lock time is set, it may lead to the inability to withdraw funds properly.

Code location: src/CyberVault.sol#L331

```
function setLockDuration(uint256 _lockDuration) external onlyOwner {
    lockDuration = _lockDuration;
}
```

Solution

It is recommended to check whether the set lockDuration value in the setLockDuration function of the CyberVault contract is the same as the lockDuration value in CyberStakingPool.

Status

Acknowledged

[N3] [Critical] CyberVault withdrawals have flaws

Category: Design Logic Audit

Content

In the CyberVault contract, it uses the ERC4626 standard to manage user funds. However, after users deposit Cyber tokens into CyberVault, the vault stakes them into CyberStakingPool to earn rewards. Since withdrawing from CyberStakingPool requires a certain lock time, when users withdraw from CyberVault, the protocol transfers the user's shares to the vault for temporary storage. Once the lock time ends, the protocol burns the previously stored shares and returns the corresponding funds to the user. During the lock period, CyberVault users' shares no longer earn rewards, but when new users deposit, the locked shares and assets still participate

in the share calculation. This means that the new rewards generated by CyberVault during the user's lock period will be implicitly allocated to the locked users when calculating the shares for new user deposits. This will cause new users to receive more shares than expected. When the locked users successfully withdraw, new users will receive more funds than expected when they withdraw.

Code location: src/CyberVault.sol#L373-L409

```
function _convertToShares(
    uint256 assets,
    Math.Rounding rounding,
    bool countFee
) private view returns (uint256) {
    uint256 totalAssets_;
    if (countFee) {
        totalAssets_ = _totalAssetsWithoutFee();
    } else {
        totalAssets_ = totalAssets();
    }
    return
        assets.mulDiv(
            totalSupply() + 10 ** _decimalsOffset(),
            totalAssets_ + 1,
            rounding
        );
}

function _convertToAssets(
    uint256 shares,
    Math.Rounding rounding,
    bool countFee
) private view returns (uint256) {
    uint256 totalAssets_;
    if (countFee) {
        totalAssets_ = _totalAssetsWithoutFee();
    } else {
        totalAssets_ = totalAssets();
    }
    return
        shares.mulDiv(
            totalAssets_ + 1,
            totalSupply() + 10 ** _decimalsOffset(),
            rounding
        );
}
```

Solution

It is recommended to subtract the locked portion of assets and shares when retrieving the totalAssets and totalSupply values.

Status

Fixed; Fixed in commit b3456712ed0bb801d88a76341d117b96181db9f7.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002406140001	SlowMist Security Team	2024.06.11 - 2024.06.14	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 critical risk, and 2 suggestions. All the findings were 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