Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.03.24, the SlowMist security team received the Bifrost team's security audit application for Bifrost vETH 2.0, 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

Serial Number	Audit Class	Audit Subclass
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
		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 covers the Vault, Core, Deposit, Token and Claim modules of Bifrost vETH 2.0.

Among them, users can make deposits in the Core module to obtain vETH 2.0 tokens and earn rewards by burning vETH 2.0 tokens through withdrawals. The operator can increase or decrease the user's reward in the Vault module, and the owner can deposit the ETH deposited by the user into the contract via the Deposit contract.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing zero value check	Design Logic Audit	Medium	Fixed
N2	The DoS issue	Denial of Service Vulnerability	Low	Fixed
N3	Lack of inequality judgment	Others	Low	Fixed
N4	Lack of event records	Others	Suggestion	Fixed
N5	Redundant code	Others	Suggestion	Ignored

4 Code Overview

4.1 Contracts Description

Audit version:

https://github.com/bifrost-finance/bifrost-vETH-2.0

commit: b6c8d412f4738a39bc2762e2fe9bd1938e79fba8

Fixed version:

https://github.com/bifrost-finance/bifrost-vETH-2.0

commit: 55bccec11409a851566a023e442259970c01b914

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:

SLPCore			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
mint	External	Payable	nonReentrant whenNotPaused
renew	External	Can Modify State	nonReentrant whenNotPaused
withdrawRequest	External	Can Modify State	nonReentrant whenNotPaused
withdrawComplete	External	Can Modify State	nonReentrant whenNotPaused
addReward	External	Can Modify State	onlyVault
removeReward	External	Can Modify State	onlyVault
depositWithdrawal	External	Payable	-
setFeeRate	External	Can Modify State	onlyOwner
setFeeReceiver	External	Can Modify State	onlyOwner
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
_setFeeRate	Private	Can Modify State	-
_setFeeReceiver	Private	Can Modify State	-
_sendValue	Private	Can Modify State	-
calculateVTokenAmount	Public	-	-
calculateTokenAmount	Public	-	-

SLPCore			
getTotalETH	Public	-	-
canWithdrawalAmount	Public	-	-

vETH2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 Pausable Ownable
mint	External	Can Modify State	onlySLPCore
burn	External	Can Modify State	onlySLPCore
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
setSLPCore	External	Can Modify State	onlyOwner

SLPDeposit			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
depositETH	External	Payable	-
batchDepositWithProof	External	Can Modify State	onlyOwner
batchDeposit	External	Can Modify State	onlyOwner
withdrawETH	External	Can Modify State	onlySLPCore
setMerkleRoot	External	Can Modify State	onlyOwner
setCredential	External	Can Modify State	onlyOwner
setSLPCore	External	Can Modify State	onlyOwner
_sendValue	Private	Can Modify State	-
innerDeposit	Private	Can Modify State	-

SLPDeposit			
checkDepositDataRoot	Public	-	-
getValidatorData	Public	-	-

vETH2Claim			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
claim	External	Can Modify State	nonReentrant

WithdrawalVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
increaseWithdrawalNode	External	Can Modify State	onlyOperator
addReward	External	Can Modify State	onlyOperator checkReward
removeReward	External	Can Modify State	onlyOperator checkReward
setSLPCore	External	Can Modify State	onlyOwner
setOperator	External	Can Modify State	onlyOwner
setRewardNumerator	External	Can Modify State	onlyOwner
getTodayTimestamp	Public	-	-

MevVault			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
addReward	External	Can Modify State	onlyOperator
setSLPCore	External	Can Modify State	onlyOwner

MevVault			
setOperator	External	Can Modify State	onlyOwner
<Receive Ether>	External	Payable	-
getTodayTimestamp	Public	-	-
getDays	Public	-	-
getReward	Public	-	-

4.3 Vulnerability Summary

[N1] [Medium] Missing zero value check

Category: Design Logic Audit

Content

In the SLPCore contract, the user can mint the vETH2 tokens by calling the mint function or renew function, and the vault can mint vETH2 as reward fee to Treasury by calling the addReward function. However, there is no check that tokenPool and totalSupply are equal to 0. In the initial case (when vETH2 tokens are not yet minted or after withdrawing all eth from the contract in the withdrawRequest function), tokenPool = IERC20Upgradeable(_vETH2).totalSupply() = 0. This will cause the calculateVTokenAmount function to fail to calculate the result correctly. This will make the entire contract unusable and fail to mint tokens.

Code Location:

contracts/SLPCore.sol

```
function mint() external payable nonReentrant whenNotPaused {
    uint256 tokenAmount = msg.value;
    require(tokenAmount > 0, "Zero amount");
    uint256 vTokenAmount = calculateVTokenAmount(tokenAmount);
    require(vTokenAmount > 0, "Zero amount");
    tokenPool = tokenPool + tokenAmount;
    ISLPDeposit(slpDeposit).depositETH{value: tokenAmount}();
    IVETH(vETH2).mint(msg.sender, vTokenAmount);

    emit Deposited(msg.sender, tokenAmount, vTokenAmount);
}
```

```
    ...

    function renew(uint256 vETH1Amount) external nonReentrant whenNotPaused {
        require(vETH1Amount > 0, "Zero amount");

        uint256 tokenAmount = vETH1Amount;
        uint256 vTokenAmount = calculateVTokenAmount(tokenAmount);
        tokenPool = tokenPool + tokenAmount;
        IERC20Upgradeable(vETH1).safeTransferFrom(msg.sender, DEAD_ADDRESS,
tokenAmount);
        IVETH(vETH2).mint(msg.sender, vTokenAmount);

        emit Renewed(msg.sender, tokenAmount, vTokenAmount);
    }


   ...

    function addReward(uint256 amount) external onlyVault {
        uint256 tokenFee = (amount * feeRate) / FEE_RATE_DENOMINATOR;
        uint256 vTokenFee = calculateVTokenAmount(tokenFee);
        tokenPool = tokenPool + amount;
        // Fee: mint vETH as reward fee to Treasury
        IVETH(vETH2).mint(feeReceiver, vTokenFee);

        emit RewardAdded(msg.sender, amount, vTokenFee);
    }
```

Solution

Add the judgment that tokenPool and totalSupply are equal to 0, and consider adding the initial case (tokenPool = totalSupply = 0) when minting tokens vETH2 tokens, the number of tokens minted is directly equal to the amount parameter or msg.value passed in.

Status

Fixed

[N2] [Low] The DoS issue

Category: Denial of Service Vulnerability

Content

The length of the external array passed in can be controlled. If the length is large, it will cause DoS because of the number of for loops.

Code location:

contracts/SLPDeposit.sol#L69-98

```
function batchDepositWithProof(
    uint256 batchId,
    bytes32[] memory proof,
    bool[] memory proofFlags,
    Validator[] memory validators
) external onlyOwner {
    bytes32 root = merkleRoots[batchId];
    require(root != bytes32(0), "Merkle root not exists");

    bytes32[] memory leaves = new bytes32[](validators.length);
    for (uint256 i = 0; i < validators.length; i++) {
        leaves[i] = keccak256(validators[i].withdrawal_credentials);
    }
    require(
        MerkleProofUpgradeable.multiProofVerify(proof, proofFlags, root, leaves),
        "Merkle proof verification failed"
    );

    for (uint256 i = 0; i < validators.length; i++) {
        innerDeposit(validators[i]);
    }
}

function batchDeposit(Validator[] calldata validators) external onlyOwner {
    require(withdrawalCredentials[0] == 0x01, "Wrong credential prefix");
    for (uint256 i = 0; i < validators.length; i++) {
        require(checkDepositDataRoot(validators[i]), "Invalid deposit data");
        innerDeposit(validators[i]);
    }
}
```

Solution

It is recommended to process in batches during the for loop or limit number of for loops to avoid DoS caused by a large number of loops.

Status

Fixed

[N3] [Low] Lack of inequality judgment

Category: Others

Content

In the SLPCore contract, When the removeReward function is called, the tokenPool is reduced according to the amount parameter passed in, but there is no determination that the tokenPool cannot equal the amount.

If the amount parameter is equal to tokenPool, then tokenPool will be reduced to zero. This will cause the calculateVTokenAmount function to not calculate the number of VToken properly.

Code Location:

contracts/SLPCore.sol#L162-166

```
    function removeReward(uint256 amount) external onlyVault {
        tokenPool = tokenPool - amount;

        emit RewardRemoved(msg.sender, amount);
    }
```

Solution

It is recommended to add a determination that the tokenPool is not equal to the amount.

Status

Fixed

[N4] [Suggestion] Lack of event records

Category: Others

Content

1.In the MevVault contract, the Owner role can set the slpCore and operator, but there is no event log.

Code Location:

contracts/MevVault.sol#L77-85

```
    function setSLPCore(address _slpCore) external onlyOwner {
        require(_slpCore != address(0), "Invalid SLP core address");
        slpCore = ISLPCore(_slpCore);
    }
```

```
    function setOperator(address _operator) external onlyOwner {
        require(_operator != address(0), "Invalid operator address");
        operator = _operator;
    }
```

2.In the SLPCore contract, the Owner role can set the feeRate and the feeReceiver, but there is no event log.

Code Location:

contracts/SLPCore.sol#L190-198

```
    function _setFeeRate(uint256 _feeRate) private {
        require(_feeRate <= FEE_RATE_DENOMINATOR, "Fee rate exceeds range");
        feeRate = _feeRate;
    }

    function _setFeeReceiver(address _feeReceiver) private {
        require(_feeReceiver != address(0), "Invalid fee receiver address");
        feeReceiver = _feeReceiver;
    }
```

3.In the SLPDeposit contract, the Owner role can set the merkleRoots, withdrawalCredentials and the slpCore,

but there is no event log.

Code Location:

contracts/SLPDeposit.sol#L104-118

```
    function setMerkleRoot(uint256 batchId, bytes32 merkleRoot) external onlyOwner {
        require(merkleRoots[batchId] == bytes32(0), "Merkle root exists");
        require(merkleRoot != bytes32(0), "Invalid merkle root");
        merkleRoots[batchId] = merkleRoot;
    }

    function setCredential(address receiver) external onlyOwner {
        require(receiver != address(0), "Invalid receiver");
        withdrawalCredentials = abi.encodePacked(bytes12(0x010000000000000000000000),
receiver);
    }

    function setSLPCore(address _slpCore) external onlyOwner {
        require(_slpCore != address(0), "Invalid SLP core address");
```

```
        slpCore = _slpCore;
    }
```

4.In the vETH2 contract, the Owner role can set the slpCore, but there is no event log.

Code Location:

contracts/vETH2.sol#L35-38

```
    function setSLPCore(address _slpCore) external onlyOwner {
        require(_slpCore != address(0), "Invalid SLP core address");
        slpCore = _slpCore;
    }
```

5.In the WithdrawalVault contract, the Owner can set the slpCore, operator and the rewardNumerator, but there is no event log.

Code Location:

contracts/WithdrawalVault.sol#L89-102

```
    function setSLPCore(address _slpCore) external onlyOwner {
        require(_slpCore != address(0), "Invalid SLP core address");
        slpCore = ISLPCore(_slpCore);
    }

    function setOperator(address _operator) external onlyOwner {
        require(_operator != address(0), "Invalid operator address");
        operator = _operator;
    }

    function setRewardNumerator(uint256 _rewardNumerator) external onlyOwner {
        require(_rewardNumerator <= REWARD_DENOMINATOR, "Reward numerator too
large");
        rewardNumerator = _rewardNumerator;
    }
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed

[N5] [Suggestion] Redundant code

Category: Others

Content

In the SLPDeposit contract, the withdrawETH function can only be called by the SLPCore contract. But the SLPCore contract does not implement the relevant logic code to call this function.

Code Location:

contracts/withdrawETH.sol#L100-102

```
    function withdrawETH(address recipient, uint256 amount) external onlySLPCore {
        _sendValue(payable(recipient), amount);
    }
```

Solution

If it is determined that it will not be used, the redundant code can be deleted.

Status

Ignored; The project team response: in order to follow the expansion of the function of the code reserved in advance, no need to delete.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002303290002	SlowMist Security Team	2023.03.24 - 2023.03.29	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 medium risk, 2 low risk, 2 suggestion vulnerabilities. And 1 suggestion vulnerability was ignored; All other findings were fixed. 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