Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.08.15, the SlowMist security team received the team's security audit application for wBETH, 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 the wBETH project that contains the warp eth to wbeth part and the unwarp wbeth to eth part. Users can deposit their eth from ETH and BSC chains to obtain the wrapped staked token wBETH and unwarp the wBETH to ETH to withdraw their assets. The withdrawal operation is to burn their wBETH and request withdrawal first. After the claim time is reached, there should be enough available allocated amount users can claim their ETH. The minting of the wrapped tokens and update of exchange rate of staked tokens is done using a rate limiting functionality. The callers executing these functionalities are configured in the RateLimit contract by the owner. The rate limiting parameters, such as the maximum allowance of the caller, current allowance, interval, and last time of setting allowance, determine how many times a caller can exercise these functionalities and how much value can be minted.

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	call() should be used instead of transfer()	Others	Suggestion	Fixed
N3	Preemptive Initialization	Race Conditions Vulnerability	Suggestion	Acknowledged
N4	Possible calculation truncation	Arithmetic Accuracy Deviation Vulnerability	Suggestion	Acknowledged
N5	Multiple Solidity versions in use	Others	Suggestion	Acknowledged
N6	Dev address setting enhancement suggestions	Others	Suggestion	Acknowledged
N7	Inability to claim due to insufficient availableAllocateAmount	Design Logic Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/earn-tech-git/wbeth/tree/develop_unwrap

commit: 279917103288e378765d50993165e8805d7e639e

Fixed Version:

https://github.com/earn-tech-git/wbeth/tree/develop_unwrap

commit: 2c9d21c8007e0af7c770a6fbf13cb5e1a6899d77

The main network address of the contract is as follows:

Contract Address (ETH)	
WBETH_TOKEN_ADDRESS	0xa2E3356610840701BDf5611a53974510Ae27E2e1
Oracle	0x81720695e43A39C52557Ce6386feB3FAAC215f06
UnwrapTokenV1ETH (Implementation)	0x542059d658624DF6452b22B10302A15a6AB59f10
UnwrapTokenProxy	0x79973d557CD9dd87eb61E250cc2572c990e20196
WrapTokenV2ETH (Implementation)	0xfe928A7D8Be9c8cEce7e97F0Ed5704f4fA2cb42A

Contract Address (BSC)	
Oracle	0x81720695e43A39C52557Ce6386feB3FAAC215f06
WBETH_TOKEN_ADDRESS	0xa2E3356610840701BDf5611a53974510Ae27E2e1
UnwrapTokenV1BSC (Implementation)	0x542059d658624DF6452b22B10302A15a6AB59f10
UnwrapTokenV1BSC (Proxy)	0x79973d557CD9dd87eb61E250cc2572c990e20196
WrapTokenV2BSC (Implementation)	0xfe928A7D8Be9c8cEce7e97F0Ed5704f4fA2cb42A

4.2 Visibility Description

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

RateLimit			
Function Name	Visibility	Mutability	Modifiers
configureCaller	External	Can Modify State	onlyOwner
removeCaller	External	Can Modify State	onlyOwner
estimatedAllowance	External	-	-
currentAllowance	Public	Can Modify State	-

RateLimit			
_replenishAllowance	Internal	Can Modify State	-
_getReplenishAmount	Internal	-	-

WrapTokenV2BSC			
Function Name	Visibility	Mutability	Modifiers
deposit	External	Can Modify State	-
supplyEth	External	Can Modify State	onlyOperator
moveToStakingAddress	External	Can Modify State	onlyOperator
_safeTransfer	Internal	Can Modify State	-
_safeTransferFrom	Internal	Can Modify State	-
requestWithdrawEth	External	Can Modify State	-
moveToUnwrapAddress	External	Can Modify State	onlyOperator
approve	External	Can Modify State	onlyOperator

StakedTokenV2			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
mint	External	Can Modify State	onlyMinters
updateOracle	External	Can Modify State	onlyOwner
updateExchangeRate	External	Can Modify State	onlyOracle
updateEthReceiver	External	Can Modify State	onlyOwner
updateOperator	External	Can Modify State	onlyOwner

StakedTokenV2			
oracle	Public	-	-
exchangeRate	Public	-	-
ethReceiver	Public	-	-
operator	Public	-	-
_mint	Internal	Can Modify State	whenNotPaused notBlacklisted notBlacklisted
_burn	Internal	Can Modify State	whenNotPaused notBlacklisted

WrapTokenV2ETH			
Function Name	Visibility	Mutability	Modifiers
deposit	External	Payable	-
supplyEth	External	Payable	onlyOperator
moveToStakingAddress	External	Can Modify State	onlyOperator
requestWithdrawEth	External	Can Modify State	-
moveToUnwrapAddress	External	Can Modify State	onlyOperator

UnwrapTokenV1ETH			
Function Name	Visibility	Mutability	Modifiers
rechargeFromRechargeAddress	External	Payable	whenNotPaused onlyRechargeAddress
moveToBackAddress	External	Can Modify State	onlyOperator
moveFromWrapContract	External	Payable	whenNotPaused onlyWrapTokenAddress
_rechargeAmount	Internal	Can Modify State	-

MintForwarder			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	onlyOwner
mint	External	Can Modify State	onlyCallers

WrapTokenV1ETH			
Function Name	Visibility	Mutability	Modifiers
deposit	External	Payable	-
supplyEth	External	Payable	onlyOperator
moveToStakingAddress	External	Can Modify State	onlyOperator

UnwrapTokenV1			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	-
requestWithdraw	External	Can Modify State	onlyWrapTokenAddress
getUserWithdrawRequests	External	-	-
getWithdrawRequests	External	-	-
claimWithdraw	External	Can Modify State	whenNotPaused notBlacklisted
allocate	External	Can Modify State	whenNotPaused onlyOperator
getNeedRechargeEthAmount	Public	-	-
_getCurrentBalance	Internal	-	-
_transferEth	Internal	Can Modify State	-
setNewOperator	External	Can Modify State	onlyOwner
setRechargeAddress	External	Can Modify State	onlyOwner

UnwrapTokenV1			
setEthBackAddress	External	Can Modify State	onlyOwner
setLockTime	External	Can Modify State	onlyOperator
setNewEthStaked	External	Can Modify State	onlyOperator
<Receive Ether>	External	Payable	-

StakedTokenV1			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
mint	External	Can Modify State	onlyMinters
updateOracle	External	Can Modify State	onlyOwner
updateExchangeRate	External	Can Modify State	onlyOracle
updateEthReceiver	External	Can Modify State	onlyOwner
updateOperator	External	Can Modify State	onlyOwner
oracle	Public	-	-
exchangeRate	Public	-	-
ethReceiver	Public	-	-
operator	Public	-	-
_mint	Internal	Can Modify State	whenNotPaused notBlacklisted notBlacklisted

WrapTokenV1BSC			
Function Name	Visibility	Mutability	Modifiers
deposit	External	Can Modify State	-

WrapTokenV1BSC			
supplyEth	External	Can Modify State	onlyOperator
moveToStakingAddress	External	Can Modify State	onlyOperator
_safeTransfer	Internal	Can Modify State	-
_safeTransferFrom	Internal	Can Modify State	-

UnwrapTokenV1BSC			
Function Name	Visibility	Mutability	Modifiers
rechargeFromRechargeAddress	External	Can Modify State	whenNotPaused onlyRechargeAddress
moveFromWrapContract	External	Can Modify State	whenNotPaused onlyWrapTokenAddress
moveToBackAddress	External	Can Modify State	onlyOperator
_rechargeAmount	Internal	Can Modify State	-
_getCurrentBalance	Internal	-	-
_transferEth	Internal	Can Modify State	-
_safeTransfer	Internal	Can Modify State	-
_safeTransferFrom	Internal	Can Modify State	-

ExchangeRateUpdater			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	onlyOwner
updateExchangeRate	Public	Can Modify State	onlyCallers

Deployer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
deploy	Public	Can Modify State	-
_isContract	Internal	-	-

UnwrapTokenProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AdminUpgradeabilityProxy

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the UnwrapTokenV1 contract, the owner role can set the operatorAddress, the rechargeAddress, the ethBackAddress, the ethStaked address, and the lockTime. Wrong configuration and sudden modification will affect the user's normal withdrawal request and claim.

Code location:

staking/UnwrapTokenV1.sol#344-388

```
function setNewOperator(address _newOperatorAddress) external onlyOwner {
    require(_newOperatorAddress != address(0), "zero address provided");
    operatorAddress = _newOperatorAddress;
    emit OperatorUpdated(_newOperatorAddress);
}

function setRechargeAddress(address _newRechargeAddress) external onlyOwner {
    require(_newRechargeAddress != address(0), "zero address provided");
    rechargeAddress = _newRechargeAddress;
    emit RechargeAddressUpdated(_newRechargeAddress);
}
```

```
    function setEthBackAddress(address _newEthBackAddress) external onlyOwner {
        require(_newEthBackAddress != address(0), "zero address provided");
        ethBackAddress = _newEthBackAddress;
        emit EthBackAddressUpdated(_newEthBackAddress);
    }

    function setLockTime(uint256 _newLockTime) external onlyOperator {
        require(_newLockTime >= MIN_LOCK_TIME, "LockTime is too small");
        lockTime = _newLockTime;
        emit LockTimeUpdated(operatorAddress, lockTime);
    }

    function setNewEthStaked(uint256 _newEthStakedAmount) external onlyOperator {
        require(ethStaked != _newEthStakedAmount, "ethStaked not change");
        ethStaked = _newEthStakedAmount;
        emit EthStakedUpdated(msg.sender, _newEthStakedAmount);
    }
```

2.In the StakedTokenV1 and StakedTokenV2 contracts, the owner role can set the minters, the Oracle address, the ethReceiver, and the Operator role. These roles can affect the amount of mint tokens, the ExchangeRate, and the withdrawal request and claim.

Code location:

staking/StakedTokenV1.sol#169-239

staking/upgrade/StakedTokenV2.sol#185-255

```
    function updateOracle(address newOracle) external onlyOwner {
        require(
            newOracle != address(0),
            "StakedTokenV1: oracle is the zero address"
        );
        require(
            newOracle != oracle(),
            "StakedTokenV1: new oracle is already the oracle"
        );
        bytes32 position = _EXCHANGE_RATE_ORACLE_POSITION;
        assembly {
            sstore(position, newOracle)
        }
        emit OracleUpdated(newOracle);
    }

    function updateExchangeRate(uint256 newExchangeRate) external onlyOracle {
        require(
            newExchangeRate >= _EXCHANGE_RATE_UNIT,
```

```
            "StakedTokenV1: new exchange rate cannot be less than 1e18"
        );
        bytes32 position = _EXCHANGE_RATE_POSITION;
        assembly {
            sstore(position, newExchangeRate)
        }
        emit ExchangeRateUpdated(msg.sender, newExchangeRate);
    }

    function updateEthReceiver(address newEthReceiver) external onlyOwner {
        require(
            newEthReceiver != address(0),
            "StakedTokenV1: newEthReceiver is the zero address"
        );
        address currentReceiver = ethReceiver();
        require(newEthReceiver != currentReceiver, "StakedTokenV1: newEthReceiver is
already the ethReceiver");
        bytes32 position = _ETH_RECEIVER_POSITION;
        assembly {
            sstore(position, newEthReceiver)
        }
        emit EthReceiverUpdated(currentReceiver, newEthReceiver);
    }

    function updateOperator(address newOperator) external onlyOwner {
        require(
            newOperator != address(0),
            "StakedTokenV1: newOperator is the zero address"
        );
        address currentOperator = operator();
        require(newOperator != currentOperator, "StakedTokenV1: newOperator is
already the operator");
        bytes32 position = _OPERATOR_POSITION;
        assembly {
            sstore(position, newOperator)
        }
        emit OperatorUpdated(currentOperator, newOperator);
    }
```

3.In the FiatTokenProxy contract, the admin role can upgrade the contract through the upgradeToAndCall and upgradeTo functions.

4.The owner of the RateLimit contract configures the allowance of the caller which decreases as when the caller mints new tokens or updates the exchange rate and it increases up to a maxAllowance parameter with a time schedule dictated by the rate limit functionality. There is a rare scenario that can happen where the allowance of

all of the callers is insufficient to update the exchange rate, and the replenishment of allowance needs a long wait time. Given the volatility of the crypto market, if the exchange rate is not updated timely, systems and protocols depending on it can be dramatically affected and the trading of the staked tokens could be impacted.

Code location:

RateLimit.sol#109-122

```
    function configureCaller(
        address caller,
        uint256 amount,
        uint256 interval
    ) external onlyOwner {
        require(caller != address(0), "RateLimit: caller is the zero address");
        require(amount > 0, "RateLimit: amount is zero");
        require(interval > 0, "RateLimit: interval is zero");
        callers[caller] = true;
        maxAllowances[caller] = allowances[caller] = amount;
        allowancesLastSet[caller] = block.timestamp;
        intervals[caller] = interval;
        emit CallerConfigured(caller, amount, interval);
    }
```

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. And the authority involving user funds should be managed by the community, and the EOA address can manage the authority involving emergency contract suspension. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged

[N2] [Suggestion] call() should be used instead of transfer()

Category: Others

Content

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas

cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase in the SLOAD instruction.

Code location:

staking/UnwrapTokenV1.sol#336-338

```
    function _transferEth(address _recipient, uint256 _ethAmount) internal virtual {
        payable(_recipient).transfer(_ethAmount);
    }
```

Solution

It is recommended to use call() instead of transfer(), but be sure to respect the CEI pattern or add re-entrancy guards, and the return value should be checked.

Status

Fixed

[N3] [Suggestion] Preemptive Initialization

Category: Race Conditions Vulnerability

Content

By calling the initialize and deploy functions to initialize the contracts, there is a potential issue that malicious attackers preemptively call the initialize function to initialize.

Code location:

staking/UnwrapTokenV1.sol#129

Deployer.sol#58

```
function initialize(
        address _newOperatorAddress,
        address _newEthBackAddress,
        address _newRechargeAddress,
        address _newPauser,
        address _newBlacklister,
        address _newOwner
    ) public {
        ......
    }
```

```
function deploy(
        address _tokenProxy,
        address _tokenImpl,

        uint256 _mintAllowance,

        address _oracleCaller,
        uint256 _allowedAmountPerInterval,
        uint256 _callerInterval
    ) public {
        ......
    }
```

Solution

It is suggested that the initialize operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker and also recommended to add the event log.

Status

Acknowledged

[N4] [Suggestion] Possible calculation truncation

Category: Arithmetic Accuracy Deviation Vulnerability

Content

In the RateLimit contract, the amountToReplenish is calculated by the division `(secondsSinceAllowanceSet * maxAllowances[caller]) / intervals[caller];`. If the value of the numerator is less than intervals[caller], this division can truncate towards 0. Since the result of the division is returned by the _getReplenishAmount function and is used in the _replenishAllowance function to update the caller's allowance, this truncation can lead to a failure in updating the caller's allowance.

Code location:

RateLimit.sol#195-196

```
        uint256 amountToReplenish = (secondsSinceAllowanceSet *
            maxAllowances[caller]) / intervals[caller];
```

Solution

It's recommended to consider setting the intervals, maxAllowances, and allowances to compatible values.

Status

Acknowledged

[N5] [Suggestion] Multiple Solidity versions in use

Category: Others

Content

Throughout the code base there are different versions of Solidity being used. Token contracts are specifically using version 0.6.12 while other contracts allow compiling with version 0.8.6.

Solution

It's recommended to allow all contracts in the code base to be compiled with the same Solidity version.

Status

Acknowledged

[N6] [Suggestion] Dev address setting enhancement suggestions

Category: Others

Content

In the can set the StakedTokenV1 and StakedTokenV2 contracts, the owner role can set the ethReceiver address to move the eth. If the address is an EOA address, in a scenario where the private keys are leaked, the team's revenue will be stolen.

Code location:

staking/StakedTokenV1.sol#205-219

staking/upgrade/StakedTokenV2.sol#221-235

```
    function updateEthReceiver(address newEthReceiver) external onlyOwner {
        require(
            newEthReceiver != address(0),
            "StakedTokenV1: newEthReceiver is the zero address"
        );

        address currentReceiver = ethReceiver();
        require(newEthReceiver != currentReceiver, "StakedTokenV1: newEthReceiver is
```

```
already the ethReceiver");

        bytes32 position = _ETH_RECEIVER_POSITION;
        assembly {
            sstore(position, newEthReceiver)
        }
        emit EthReceiverUpdated(currentReceiver, newEthReceiver);
    }
```

Solution

It is recommended to set the insurance address as a multi-signature contract to avoid the leakage of private keys and the theft of team rewards.

Status

Acknowledged

[N7] [Suggestion] Inability to claim due to insufficient availableAllocateAmount

Category: Design Logic Audit

Content

In the WrapTokenV2ETH and WrapTokenV2BSC contract, users can call the `requestWithdrawEth` function to burn their wbeth to withdraw their unwrap_ETH tokens. In this function, the withdraw operation is executed by the UnwrapTokenV1 `requestWithdraw` function. And in the requestWithdraw function, the `_currentIndex` value will be increased by the `nextIndex++` self-increment. Once the `availableAllocateAmount` is less than the `_ethAmount` or the `startAllocatedEthIndex` is not equal to the `currentIndex`, the if judgment will pass to execute the else part only, and the `startAllocatedEthIndex` will not self-increment. This can lead to the allocation failing that users can not call the claimWithdraw function to withdraw their eth. Only in the UnwrapTokenV1 contract, the operator role can call the allocate function to allocate availableAllocateAmount of ethAmount to make the `startAllocatedEthIndex++` self-increment to match the if judgment and the claimWithdraw's allocated value will be set to true.

Code location:

staking/UnwrapTokenV1.sol#187

```
    function requestWithdraw(address _recipient, uint256 _wbethAmount, uint256
_ethAmount)
        external onlyWrapTokenAddress {
```

```
        ......
        uint256 _currentIndex = nextIndex++;
        bool _allocated = false;
        if (availableAllocateAmount >= _ethAmount && startAllocatedEthIndex ==
_currentIndex) {
            _allocated = true;
            availableAllocateAmount = availableAllocateAmount.sub(_ethAmount);
            startAllocatedEthIndex++;
        } else {
            needEthAmount = needEthAmount.add(_ethAmount);
        }
        ......
    }

    function allocate(uint256 _maxAllocateNum) external whenNotPaused onlyOperator
returns (uint256)
    {
        ......
        for (uint256 _reqCount = 0; _reqCount < _maxAllocateNum &&
startAllocatedEthIndex < nextIndex &&
withdrawRequests[startAllocatedEthIndex].ethAmount <= availableAllocateAmount;
            _reqCount++
        ) {
           ......
            startAllocatedEthIndex++;
        }
        ......
    }
```

Solution

It is recommended to provide enough availableAllocateAmount to allow users to claim their own assets instead of being allocated by the operator.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002308210001	SlowMist Security Team	2023.08.15 - 2023.08.21	Medium Risk

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 and 6 suggestions. All the findings were fixed or acknowledged. The code was not fully 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