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 2023.02.03, the SlowMist security team received the team's security audit application for LightDAO, 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

The LightDAO is a decentralised autonomous organisation (DAO) that drives and governs the development and growth of the $HOPE stablecoin and the HOPE Ecosystem through community governance, liquidity provision and revenue distribution. As a decentralised autonomous organisation, the DAO's decisions and operations are jointly participated and managed by community members.

Under the LightDAO, community members can participate in the governance of the DAO and receive LT tokens as rewards. Through governance voting, community members can make decisions on important matters of the ecosystem, such as changing the protocol parameters, proposing and voting for the adoption of protocol

upgrades, allocating community funds, etc. By vote-locking LT tokens, community members can gain voting rights and reward boosts, including liquidity mining rewards and a share of transaction fees.

The DAO's smart contracts offer several modules including Agent, Staking, Gombocs(previous codename for Gauge), Controller, Minter, Token and VotingEscrow. These modules provide a complete Decentralised Finance (DeFi) ecosystem for LightDAO, enabling community members to participate in multiple aspects of decision making and operations such as liquidity provision, governance and revenue distribution.

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	The DoS issue	Denial of Service Vulnerability	Low	Fixed
N3	Missing event records	Others	Suggestion	Fixed
N4	Missing zero address check	Others	Suggestion	Fixed
N5	Missing return value check	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

Codebase:

https://github.com/Light-Ecosystem/light-dao

Audit Version:

https://github.com/Light-Ecosystem/light-dao

commit: d5ec21af35a0afc2cc4bb85a0131a11cad2b0bf5

https://github.com/Light-Ecosystem/light-lib

commit: 4765138a38d189a484d635b05cec850a7b7a03e6

Fixed Version:

https://github.com/Light-Ecosystem/light-dao

commit: eb70734264b46194eb3f7f11335a01588298c3cc

https://github.com/Light-Ecosystem/light-lib

commit: 123a60b556926b8cd5413de772f4a3dcc65bb1c1

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:

AgentManager			
Function Name	Visibility	Mutability	Modifiers
getMaxCredit	Public	-	-
getRemainingCredit	Public	-	-
isMinable	Public	-	-
isBurnable	Public	-	-
getEffectiveBlock	Public	-	-
getExpirationBlock	Public	-	-
hasAgent	Public	-	-
grantAgent	Public	Can Modify State	onlyOwner
_grantAgent	Internal	Can Modify State	-

AgentManager			
revokeAgent	Public	Can Modify State	onlyOwner
_revokeAgent	Internal	Can Modify State	-
changeEffectiveBlock	Public	Can Modify State	onlyOwner
changeExpirationBlock	Public	Can Modify State	onlyOwner
switchMinable	Public	Can Modify State	onlyOwner
switchBurnable	Public	Can Modify State	onlyOwner
increaseCredit	Public	Can Modify State	onlyOwner
decreaseCredit	Public	Can Modify State	onlyOwner
_increaseRemainingCredit	Internal	Can Modify State	-
_decreaseRemainingCredit	Internal	Can Modify State	-

HOPESalesAgent			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
buy	External	Can Modify State	whenNotPaused
remainingCredit	External	-	-
balanceOf	External	-	-
redeem	External	Can Modify State	onlyOwner whenNotPaused
addCurrency	External	Can Modify State	onlyOwner
changeCurrencyRate	External	Can Modify State	onlyOwner
deleteCurrency	External	Can Modify State	onlyOwner
_getToValue	Internal	-	-
pause	Public	Can Modify State	onlyOwner

HOPESalesAgent			
unpause	Public	Can Modify State	onlyOwner

LT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
__updateMiningParameters	Internal	Can Modify State	-
updateMiningParameters	External	Can Modify State	-
futureEpochTimeWrite	External	Can Modify State	-
__availableSupply	Internal	-	-
availableSupply	External	-	-
mintableInTimeframe	External	-	-
setMinter	External	Can Modify State	onlyOwner
mint	External	Can Modify State	-
burn	External	Can Modify State	-

HOPE			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
mint	Public	Can Modify State	onlyAgent onlyMinable
burn	External	Can Modify State	onlyAgent onlyBurnable
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-

HOPE			

Minter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mint	External	Can Modify State	-
mintMany	External	Can Modify State	-
mintFor	External	Can Modify State	-
toggleApproveMint	External	Can Modify State	-
_mintFor	Internal	Can Modify State	-

StakingHOPE			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
staking	External	Can Modify State	-
unstaking	External	Can Modify State	-
unstakingBalanceOf	Public	-	-
unstakingTotal	Public	-	-
unstakedBalanceOf	Public	-	-
unstakedTotal	External	-	-
redeemAll	External	Can Modify State	-
redeemByMaxIndex	External	Can Modify State	-
lpBalanceOf	Public	-	-
lpTotalSupply	Public	-	-

StakingHOPE			
transfer	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-

VotingEscrow			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getUserPointHistory	External	-	-
getLastUserSlope	External	-	-
userPointHistoryTs	External	-	-
lockedEnd	External	-	-
_checkpoint	Internal	Can Modify State	-
_depositFor	Internal	Can Modify State	-
checkpointSupply	Public	Can Modify State	-
createLockFor	External	Can Modify State	onlyOwner
createLock	External	Can Modify State	-
_createLock	Internal	Can Modify State	nonReentrant
increaseAmount	External	Can Modify State	-
increaseAmountFor	External	Can Modify State	-
_increaseAmount	Internal	Can Modify State	nonReentrant
increaseUnlockTime	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
_findBlockEpoch	Internal	-	-
balanceOf	External	-	-

VotingEscrow			
balanceOfAtTime	External	-	-
balanceOfAt	External	-	-
_supplyAt	Internal	-	-
totalSupply	External	-	-
totalSupplyAtTime	External	-	-
totalSupplyAt	External	-	-
setSmartWalletChecker	External	Can Modify State	onlyOwner
_assertNotContract	Internal	-	-

PoolGomboc			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
_deposit	Private	Can Modify State	-
deposit	External	Can Modify State	nonReentrant
deposit	External	Can Modify State	nonReentrant
deposit	External	Can Modify State	nonReentrant
_withdraw	Private	Can Modify State	-
withdraw	External	Can Modify State	nonReentrant
withdraw	External	Can Modify State	nonReentrant
_transfer	Internal	Can Modify State	-
transfer	External	Can Modify State	nonReentrant
transferFrom	External	Can Modify State	nonReentrant

PoolGomboc			
_approve	Internal	Can Modify State	-
approve	External	Can Modify State	-
increaseAllowance	External	Can Modify State	-
decreaseAllowance	External	Can Modify State	-
addReward	External	Can Modify State	onlyOwner
setRewardDistributor	External	Can Modify State	-
depositRewardToken	External	Payable	nonReentrant
_checkpointRewards	Internal	Can Modify State	-
claimedReward	External	-	-
claimableReward	External	-	-
setRewardsReceiver	External	Can Modify State	-
_claimRewards	Private	Can Modify State	-
claimRewards	External	Can Modify State	nonReentrant
claimRewards	External	Can Modify State	nonReentrant
claimRewards	External	Can Modify State	nonReentrant
lpBalanceOf	Public	-	-
lpTotalSupply	Public	-	-

AbsExternalLTRewardDistributor			
Function Name	Visibility	Mutability	Modifiers
_init	Internal	Can Modify State	-
refreshGombocRewards	External	Can Modify State	-
_setGombocAddress	Internal	Can Modify State	-

GombocController			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
gombocTypes	External	-	-
addGomboc	External	Can Modify State	onlyOwner
checkpoint	External	Can Modify State	-
checkpointGomboc	External	Can Modify State	-
gombocRelativeWeight	External	-	-
gombocRelativeWeightWrite	External	Can Modify State	-
addType	External	Can Modify State	onlyOwner
changeTypeWeight	External	Can Modify State	onlyOwner
changeGombocWeight	External	Can Modify State	onlyOwner
voteForGombocWeights	External	Can Modify State	-
getGombocWeight	External	-	-
getTypeWeight	External	-	-
getTotalWeight	External	-	-
getWeightsSumPreType	External	-	-
_getTypeWeight	Internal	Can Modify State	-
_getSum	Internal	Can Modify State	-
_getTotal	Internal	Can Modify State	-
_getWeight	Internal	Can Modify State	-
_gombocRelativeWeight	Internal	-	-
_changeGombocWeight	Internal	Can Modify State	-
_changeTypeWeight	Internal	Can Modify State	-

GombocController			
_int128ToUint256	Internal	-	-

GombocFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
createPool	External	Can Modify State	onlyOwner
allPoolsLength	External	-	-

AbsGomboc			
Function Name	Visibility	Mutability	Modifiers
_init	Internal	Can Modify State	-
_updateLiquidityLimit	Internal	Can Modify State	-
_checkpoint	Internal	Can Modify State	-
userCheckpoint	External	Can Modify State	-
claimableTokens	External	Can Modify State	-
kick	External	Can Modify State	-
integrateCheckpoint	External	-	-
setKilled	External	Can Modify State	onlyOwner
lpBalanceOf	Public	-	-
lpTotalSupply	Public	-	-

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the HOPESalesAgent contract, the owner can add currency that can buy HOPE token、delete the currency and change the currency exhange rate. If the owner's privileges are lost, it could lead to a contract being maliciously added with a worthless currency and used it to unintentionally buy large amounts of hope tokens.

Code location:

contracts/agents/HOPESalesAgent.sol#L120-165

```
function addCurrency(string memory symbol, address token, uint256 rate) external
onlyOwner {
        Currency storage cy = currencys[symbol];
        require(cy.token == address(0), "HS002");
        require(rate > 0, "HS003");
        require(token != address(0), "CE000");

        IERC20Metadata erc20 = IERC20Metadata(token);
        string memory erc20Symbol = erc20.symbol();
        require(StringUtils.hashCompareWithLengthCheck(symbol, erc20Symbol),
"HS004");
        cy.symbol = symbol;
        cy.token = token;
        cy.rate = rate;

        emit AddCurrency(symbol, token, rate);
    }
    ...
    function changeCurrencyRate(string memory symbol, uint256 rate) external
onlyOwner {
        Currency storage cy = currencys[symbol];
        require(cy.token != address(0), "CE001");
        require(rate > 0, "HS003");
        uint256 oldRate = cy.rate;
        cy.rate = rate;

        emit ChangeCurrencyRate(symbol, rate, oldRate);
    }

    ...
    function deleteCurrency(string memory symbol) external onlyOwner {
        Currency storage cy = currencys[symbol];
        require(cy.token != address(0), "CE001");
        require(this.balanceOf(symbol) == 0, "HS005");
```

```
        delete currencys[symbol];

        emit DeleteCurrency(symbol);
    }
```

2.In the GombocController contract, the owner can change the weight of gomboc and the gomboc type weight.

If the owner's privileges are lost, it will cause the user's weight calculation to be affected.

Code location:

contracts/GombocController.sol#L205-218

```
    function changeTypeWeight(int128 typeId, uint256 weight) external override
onlyOwner {
        _changeTypeWeight(typeId, weight);
    }
    ...
    function changeGombocWeight(address gombocAddress, uint256 weight) external
override onlyOwner {
        int128 gombocType = _gombocTypes[gombocAddress] - 1;
        require(gombocType >= 0, "GC000");
        _changeGombocWeight(gombocAddress, weight);
    }
```

3.The owner role can change the agent's expiration date, mint status and credit limit. If the owner's privileges are lost, it may affect the ability of users to get tokens through agents.

Code location:

contracts/agents/AgentManager.sol#L165-231

```
    function changeEffectiveBlock(address account, uint256 effectiveBlock) public
override onlyOwner {
        ...
    }

    function changeExpirationBlock(address account, uint256 expirationBlock) public
override onlyOwner {
        ...
    }

    /**
     * @dev Change the minable status of the address agent
     */
```

```
    function switchMinable(address account, bool minable) public override onlyOwner {
        ...
    }

    /**
     * @dev Change the burnable status of the address agent
     */
    function switchBurnable(address account, bool burnable) public override onlyOwner
{
        ...
    }

    /**
     * @dev Increase credit of the address agent
     * @dev After increase credit, the max credit and the remaining credit increase
simultaneously
     */
    function increaseCredit(address account, uint256 credit) public override
onlyOwner {
        ...
    }

    /**
     * @dev Decrease credit of the address agent
     * @dev After decrease credit, the max credit and the remaining credit decrease
simultaneously
     */
    function decreaseCredit(address account, uint256 credit) public override
onlyOwner {
        ...
    }
```

Solution

It is recommended to hand over the Owner role to the community or timeLock contract governance, at least multi-signature should be used.

Status

Acknowledged; The project team responded: For this issue, we currently use multi-signature wallets, which will be gradually transferred to the DAO for management.

[N2] [Low] The DoS issue

Category: Denial of Service Vulnerability

Content

The user can pass in the gombocAddressList array through the mintMany function to mint the LT. If the length of gombocAddressList is large, it will cause DoS because of the number of for loops.

Code location:

contracts/Minter.sol#L48-55

```
function mintMany(address[] memory gombocAddressList) external {
    for (uint256 i = 0; i < gombocAddressList.length; i++) {
        if (gombocAddressList[i] == address(0)) {
            continue;
        }
        _mintFor(gombocAddressList[i], msg.sender);
    }
}
```

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] [Suggestion] Missing event records

Category: Others

Content

1.In the Minter contract, the caller can toggle the approval status for mintingUser, but there are no event logs.

Code location:

contracts/Minter.sol#L73-76

```
function toggleApproveMint(address mintingUser) external {
    bool flag = allowedToMintFor[mintingUser][msg.sender];
    allowedToMintFor[mintingUser][msg.sender] = !flag;
}
```

2.In the VotingEscrow contract, the owner can set an external contract to check for approved smart contract wallets, but there are no event logs.

```
Code location:
contracts/VotingEscrow.sol#L716-718
    function setSmartWalletChecker(address _check) external onlyOwner {
        smartWalletChecker = _check;
    }
```

3.In the PoolGomboc contract, the owner can add new reward tokens and set the distributor of the reward, but there are no event logs.

Code location:

contracts/gombocs/PoolGomboc.sol#L318-333

```
    function addReward(address _rewardToken, address _distributor) external onlyOwner
{
        ...
        rewardData[_rewardToken].distributor = _distributor;
        rewardTokens[_rewardCount] = _rewardToken;
        rewardCount = _rewardCount + 1;
    }

    function setRewardDistributor(address _rewardToken, address _distributor)
external {
        ...
        rewardData[_rewardToken].distributor = _distributor;
    }
```

Solution

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

Status

Fixed

[N4] [Suggestion] Missing zero address check

Category: Others

Content

When modifying important addresses in the contract, it is not checked whether the incoming address is a zero address.

Code location:

contracts/gomboc/AbsGomboc.sol#L76-89

```
    function _init(address _lpAddr, address _minter) internal {
        require(!_initialized, "Initializable: contract is already initialized");
        _initialized = true;

        lpToken = _lpAddr;
        minter = IMinter(_minter);
        address _ltToken = minter.token();
        ltToken = ILT(_ltToken);
        controller = IGombocController(minter.controller());
        votingEscrow = IVotingEscrow(controller.votingEscrow());
        periodTimestamp[0] = block.timestamp;
        inflationRate = ltToken.rate();
        futureEpochTime = ltToken.futureEpochTimeWrite();
    }
```

Code location:

contracts/gomboc/AbsExternalLTRewardDistributor.sol#L29-35

```
    function _init(address stHopeGomboc, address minter, address ltToken) internal {
        require(!_initialized, "Initializable: contract is already initialized");
        _initialized = true;
        _stHopeGomboc = stHopeGomboc;
        _minter = minter;
        _ltToken = ltToken;
    }
```

Solution

It is recommended to add a zero address check.

Status

Fixed

[N5] [Suggestion] Missing return value check

Category: Others

Content

When transferring ERC20 tokens, the return value after the transfer is not checked. If return false, the logical should be reverted.

Code location:

contracts/gombocs/PoolGomboc.sol#L179

```
function _withdraw(uint256 _value, bool _claimRewards_) private {
    _checkpoint(msg.sender);

    if (_value != 0) {
        ...
        IERC20Metadata(lpToken).transfer(msg.sender, _value);
    }

    emit Withdraw(msg.sender, _value);
    emit Transfer(msg.sender, address(0), _value);
}
```

Solution

It is recommended to check the return value after the transfer.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002302160001	SlowMist Security Team	2023.02.03 - 2023.02.16	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, 1 low risk, 3 suggestion vulnerabilities. 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