Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.02.20, the SlowMist security team received the HOPE team's security audit application for LightDAO 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 includes two parts: the fee collection and distribution for LightDAO and a decentralized exchange.

This decentralized exchange is based on Uniswap V2, with the addition of modifiable fees and the ability to transfer rewards for stHOPE shares held by pair to the gomboc(Product name:Gauge) contracts for distribution.

Light exchange contracts have the capability to charge an “admin fee”, claimable by the feeToVault. The admin fee is represented as a percentage of the total fee collected on a swap.

For exchanges the fee is taken in the output currency and calculated against the final amount received. For example, if swapping from USDT to USDC, the fee is taken in USDC.

In addition to Swap, the fee generated by other protocols(not released) of the LightDao will also be collected and distribution.

Admin fee for all protocols of lightDao are indirectly owned by the LightDAO via a feeToVault ownership contract. This contract includes functionality to withdraw the fees, convert them to HOPE, and forward 50% them into the fee distributor contract,and forward 50% them into the gomboc(Product name:Gauge) fee distributor contract. Collectively, this process is referred to as “burning”.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of external manipulation	Unsafe External Call Audit	Critical	Fixed
N2	The DoS issue	Denial of Service Vulnerability	Low	Fixed
N3	Token compatibility issue	Others	Low	Fixed
N4	Missing slippage check	Reordering Vulnerability	Low	Fixed
N5	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Fixed
N6	Missing zero address check	Others	Suggestion	Fixed
N7	Redundant code	Others	Suggestion	Fixed
N8	Missing event records	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

Codebase:

Audit version:

https://github.com/Light-Ecosystem/swap-core

commit: d8dee3ba78b4b13d7edc851735c9fc1db65832c9

https://github.com/Light-Ecosystem/swap-periphery

commit: d763423da6bf3be69b6b6e9f9450134d99e99258

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

commit: 490654d807b7370a7bcd546d47e52a9f971782b8

Fixed Version:

https://github.com/Light-Ecosystem/swap-core

commit: 6f6413fe5893f0d32ec295b028b43bef34662c68

https://github.com/Light-Ecosystem/swap-periphery

commit: 40a540bf2f673bbd1d5fc2c1e22ce7a03a0f2571

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

commit: 28214b54d78e3d51808160b67dd546afa79a474a

The main network address of the contract is as follows:

BurnerManager: 0x0d80a72c9F98e5b1C7Fb3e3dC4d58aecA5966066

FeeDistributor: 0x99040c96bb8D931c29b2a9B91Dcfcd36162BB697

GaugeFeeDistributor: 0xE0530d1261802eb32908b72574F9a6362C898a84

UnderlyingBurner: 0x77b3CFCd79F8030fAc267da519a1D8e4F6ee5F29

SwapFeeToVault: 0xdA9C43a13A82b0D0292cF38E18Fa71a0a9F42f23

HopeSwapBurner: 0xD32864fF55Aa1ceABACe9D3e57bB113461ACea42

ApprovedTokenManager: 0x2Ca1F5e429D423701052222E598E036016173c78

UniswapV2Factory : 0x26F53fbADeEb777fb2A122dC703433d79241b64e

UniswapV2Router02 : 0x219Bd2d1449F3813c01204EE455D11B41D5051e9

HOPE-USDT Pair : 0x1c2aD915CD67284cDbc04507B11980797CF51b22

HOPE-USDT Gauge: 0x306eC09d53f4E9d86F53f6Ed412447b13cC8BB83

LT-USDT Pair: 0xA9aD6A54459635A62e883dC99861c1a69Cf2c5B3

LT-USDT Gauge: 0x99f380C31f647F23E87E5518187f172bCa6d7308

4.2 Visibility Description

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

UniswapV2Factory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
allPairsLength	External	-	-
createPair	External	Can Modify State	-
setFeeTo	External	Can Modify State	-
setFeeToSetter	External	Can Modify State	-
setApprovedTokenManager	External	Can Modify State	-

UniswapV2Pair			
Function Name	Visibility	Mutability	Modifiers
getReserves	Public	-	-
_safeTransfer	Private	Can Modify State	-
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	-
_update	Private	Can Modify State	-
_mintFee	Private	Can Modify State	-
mintFee	External	Can Modify State	lock
mint	External	Can Modify State	lock

UniswapV2Pair			
burn	External	Can Modify State	lock
swap	External	Can Modify State	lock
skim	External	Can Modify State	lock
sync	External	Can Modify State	lock
claimLightReward	External	Can Modify State	-

UniswapV2ERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_approve	Private	Can Modify State	-
_transfer	Private	Can Modify State	-
approve	External	Can Modify State	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
permit	External	Can Modify State	-

Ownable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
owner	Public	-	-
_checkOwner	Internal	-	-
renounceOwnership	Public	Can Modify State	onlyOwner

Ownable			
transferOwnership	Public	Can Modify State	onlyOwner
_transferOwnership	Internal	Can Modify State	-

Context			
Function Name	Visibility	Mutability	Modifiers
_msgSender	Internal	-	-
_msgData	Internal	-	-

ApprovedTokenManager			
Function Name	Visibility	Mutability	Modifiers
isApprovedToken	Public	-	-
approveToken	Public	Can Modify State	onlyOwner

UniswapV2Migrator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
migrate	External	Can Modify State	-

UniswapV2Router01			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
_addLiquidity	Private	Can Modify State	-

UniswapV2Router01			
addLiquidity	External	Can Modify State	ensure
addLiquidityETH	External	Payable	ensure
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-
_swap	Private	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

UniswapV2Router02			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-

UniswapV2Router02			
_addLiquidity	Internal	Can Modify State	-
addLiquidity	External	Can Modify State	ensure
addLiquidityETH	External	Payable	ensure
removeLiquidity	Public	Can Modify State	ensure
removeLiquidityETH	Public	Can Modify State	ensure
removeLiquidityWithPermit	External	Can Modify State	-
removeLiquidityETHWithPermit	External	Can Modify State	-
removeLiquidityETHSupportingFeeOnTransferTokens	Public	Can Modify State	ensure
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	Can Modify State	-
_swap	Internal	Can Modify State	-
swapExactTokensForTokens	External	Can Modify State	ensure
swapTokensForExactTokens	External	Can Modify State	ensure
swapExactETHForTokens	External	Payable	ensure
swapTokensForExactETH	External	Can Modify State	ensure
swapExactTokensForETH	External	Can Modify State	ensure
swapETHForExactTokens	External	Payable	ensure
_swapSupportingFeeOnTransferTokens	Internal	Can Modify State	-
swapExactTokensForTokensSupportingFeeOnTransferTokens	External	Can Modify State	ensure
swapExactETHForTokensSupportingFeeOnTr	External	Payable	ensure

UniswapV2Router02			
ansferTokens			
swapExactTokensForETHSupportingFeeOnTransferTokens	External	Can Modify State	ensure
quote	Public	-	-
getAmountOut	Public	-	-
getAmountIn	Public	-	-
getAmountsOut	Public	-	-
getAmountsIn	Public	-	-

BurnerManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setBurner	External	Can Modify State	onlyOwner
setManyBurner	External	Can Modify State	onlyOwner
_setBurner	Internal	Can Modify State	-

FeeDistributor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
checkpointToken	External	Can Modify State	-
_checkpointToken	Internal	Can Modify State	-
veForAt	External	-	-
vePrecentageForAt	External	-	-
checkpointTotalSupply	External	Can Modify State	-

FeeDistributor			
_checkpointTotalSupply	Internal	Can Modify State	-
_findTimestampEpoch	Internal	-	-
_findTimestampUserEpoch	Internal	-	-
_claim	Internal	Can Modify State	-
_getPointBalanceOf	Internal	-	-
claim	External	Can Modify State	whenNotPaused
claimableToken	External	Can Modify State	-
claimMany	External	Can Modify State	whenNotPaused
burn	External	Can Modify State	whenNotPaused
toggleAllowCheckpointToken	External	Can Modify State	onlyOwner
recoverBalance	External	Can Modify State	onlyOwner
setEmergencyReturn	External	Can Modify State	onlyOwner
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
stakingHOPEAndTransfer2User	Internal	Can Modify State	-

GombocFeeDistributor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
checkpointToken	External	Can Modify State	-
_checkpointToken	Internal	Can Modify State	-
veForAt	External	-	-

GombocFeeDistributor			
vePrecentageForAt	External	-	-
gombocBalancePreWeek	External	-	-
_findTimestampUserEpoch	Internal	-	-
_claim	Internal	Can Modify State	-
claim	External	Can Modify State	whenNotPaused
claimableTokens	External	Can Modify State	whenNotPaused
claimMany	External	Can Modify State	whenNotPaused
claimManyGomboc	External	Can Modify State	whenNotPaused
claimableTokenManyGomboc	External	Can Modify State	whenNotPaused
burn	External	Can Modify State	whenNotPaused
toggleAllowCheckpointToken	External	Can Modify State	onlyOwner
recoverBalance	External	Can Modify State	onlyOwner
setEmergencyReturn	External	Can Modify State	onlyOwner
stakingHOPEAndTransfer2User	Internal	Can Modify State	-
_getPointBalanceOf	Internal	-	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner

LightSwapBurner			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setRouters	External	Can Modify State	onlyOwner
burn	External	Can Modify State	-

SwapFeeToVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
withdrawAdminFee	External	Can Modify State	whenNotPaused
withdrawMany	External	Can Modify State	whenNotPaused
_burn	Internal	Can Modify State	-
burn	External	Can Modify State	-
burnMany	External	Can Modify State	-
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner

UnderlyingBurner			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
transferHopeToFeeDistributor	External	Can Modify State	whenNotPaused
recoverBalance	External	Can Modify State	onlyOwner
setRouters	External	Can Modify State	onlyOwner
burn	External	Can Modify State	-
setEmergencyReturn	External	Can Modify State	onlyOwner
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Critical] Risk of external manipulation

Category: Unsafe External Call Audit

Content

In the UniswapV2Pair contract, anyone can call the claimLightReward function to claim and deposit the reward belonging to the pair. But the incoming address parameters (lightGomboc, minter, lightToken, gomboc) are externally manipulable, the attacker can construct a malicious external contract (the malicious gomboc contract) to steal the rewards that belong to the pair.

Code location:

swap-core/contracts/UniswapV2Pair.sol#L211-225

```
    function claimLightReward(address lightGomboc, address minter, address
lightToken, address gomboc) external {
        require(lightGomboc != address(0), "Light: INVALID_LIGHT_GOMBOC");
        require(minter != address(0), "Light: INVALID_MINTER");
        require(lightToken != address(0), "Light: INVALID_LIGHT");
        require(gomboc != address(0), "Light: INVALID_GOMBOC");

        uint256 amount = ILightGomboc(lightGomboc).claimableTokens(address(this));
        require(amount>0, "Light: NO_BALANCE");
        IMinter(minter).mint(lightGomboc);

        IERC20(lightToken).approve(lightGomboc, amount);
        ILightGomboc(gomboc).depositRewardToken(lightToken, amount);

        emit ClaimLightReward(amount);
    }
```

Solution

Define the parameters to be passed in the initialization or set by the owner in the factory.

Status

Fixed

[N2] [Low] The DoS issue

Category: Denial of Service Vulnerability

Content

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

Code location:

light-dao/contracts/feeDistributor/feeDistributor.sol#L384-416

```
    function claimMany(address[] memory _receivers) external whenNotPaused returns
(uint256) {
        ...

        for (uint256 i = 0; i < _receivers.length; i++) {
            address addr = _receivers[i];
            if (addr == address(0)) {
                break;
            }
            uint256 amount = _claim(addr, votingEscrow, _lastTokenTime);
            if (amount != 0) {
                stakingHOPEAndTransfer2User(addr, amount);
                total += amount;
            }
        }

        ...
    }
```

light-dao/contracts/feeDistributor/GombocFeeDistributor.sol#L348-413

```
    function claimMany(address gomboc, address[] memory _receivers) external
whenNotPaused returns (uint256) {
        ...

        for (uint256 i = 0; i < _receivers.length; i++) {
            address addr = _receivers[i];
            if (addr == address(0)) {
                break;
            }
            uint256 amount = _claim(gomboc, addr, _lastTokenTime);
            if (amount != 0) {
                stakingHOPEAndTransfer2User(addr, amount);
                total += amount;
            }
        }

        ...
    }
```

```
    function claimManyGomboc(address[] memory gombocList, address receiver) external
whenNotPaused returns (uint256) {

        ...

        for (uint256 i = 0; i < gombocList.length; i++) {
            address gomboc = gombocList[i];
            if (gomboc == address(0)) {
                break;
            }
            uint256 amount = _claim(gomboc, receiver, _lastTokenTime);
            if (amount != 0) {
                total += amount;
            }
        }

        ...
    }
```

light-dao/contracts/feeDistributor/SwapFeeToVault.sol#L38-42

```
    function withdrawMany(address[] memory pools) external whenNotPaused {
        for (uint256 i = 0; i < pools.length; i++) {
            this.withdrawAdminFee(pools[i]);
        }
    }
```

light-dao/contracts/feeDistributor/SwapFeeToVault.sol#L59-63

```
    function burnMany(IERC20[] calldata tokens) external {
        for (uint i = 0; i < tokens.length; i++) {
            _burn(tokens[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] Token compatibility issue

Category: Others

Content

In the burn function of the LightSwapBurner contract, it will transfer the specified token into this contract through the transferFrom function. If the specified token is a deflationary token, the actual number of the token received by the contract is less than the value of the amount parameter passed in by the user. This may cause the subsequent swap operation to fail.

Code location:

light-dao/contracts/feeDistributor/LightSwapBurner.sol#L37-66

```
    function burn(address to, IERC20 token, uint amount) external {
        if (token == HOPE) {
            require(token.transferFrom(msg.sender, to, amount), "LSB00");
            return;
        }

        require(token.transferFrom(msg.sender, address(this), amount), "LSB01");

        ISwapRouter bestRouter = routers[0];
        uint bestExpected = 0;
        address[] memory path = new address[](2);
        path[0] = address(token);
        path[1] = address(HOPE);

        for (uint i = 0; i < routers.length; i++) {
            uint[] memory expected = routers[i].getAmountsOut(amount, path);
            if (expected[0] > bestExpected) {
                bestExpected = expected[0];
                bestRouter = routers[i];
            }
        }

        if (!approved[bestRouter][token]) {
            bool success = IERC20(token).approve(address(bestRouter),
type(uint).max);
            require(success, "LSB01");
            approved[bestRouter][token] = true;
        }
```

```
        bestRouter.swapExactTokensForTokens(amount, 0, path, to, block.timestamp);
    }
```

Solution

It is recommended to take the difference between the token balance of the contract before and after the user's transfer as the actual amount transferred by the user.

Status

Fixed

[N4] [Low] Missing slippage check

Category: Reordering Vulnerability

Content

In the LightSwapBurner and UnderlyingBurner contract, when performing the burn operation, the swapExactTokensForTokens function will be used to exchange tokens. However, the swapExactTokensForTokens function does not perform a slippage check, which will result in a high probability of being subjected to a sandwich attack by MEV Bot when performing the above operations. This will result in far fewer exchanges than expected, or even very few left.

Code location:

light-dao/contracts/feeDistributor/LightSwapBurner.sol#L37-66

```
    function burn(address to, IERC20 token, uint amount) external {
        ...

        bestRouter.swapExactTokensForTokens(amount, 0, path, to, block.timestamp);
    }
```

light-dao/contracts/feeDistributor/UnderlyingBurner.sol#L104-128

```
    function burn(IERC20Upgradeable token, uint amount) external {
        ...

        bestRouter.swapExactTokensForTokens(amount, 0, path, to, block.timestamp);
    }
```

Solution

It is recommended to add a check for slippage to ensure that amountOutMin meets user expectations.

Status

Fixed; The project team's response: For the logic of burn, we refer to the logic of curve. They do not have a sliding point check. Combined with the business, I guess it is because the actual business scenario will carry out the burnMany operation. If each of them needs to add a sliding point check, the whole burn process will be very complex.

[N5] [Low] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the UniswapV2Factory contract, the feeToSetter role can set the feeRateNumerator through the setFeeRateNumerator function. However, there is no limit to the feeRateNumerator parameter that will be passed in, which can lead to high user fees if it is modified too much.

Code location:

swap-periphery/contracts/UniswapV2Factory.sol#L74-78

```
    function setFeeRateNumerator(address tokenA, address tokenB, uint32
feeRateNumerator) external {
        require(msg.sender == feeToSetter, 'HopeSwap: FORBIDDEN');
        IUniswapV2Pair(getPair[tokenA]
[tokenB]).setFeeRateNumerator(feeRateNumerator);
        emit SetFeeRateNumerator(tokenA, tokenB, feeRateNumerator);
    }
```

Solution

It is recommended that the feeToSetter role should be handed over to community or timeLock contract governance, that at least multiple signatures be used, and that a suitable upper and lower range be set for the feeRateNumerator, e.g. 0 to 1000.

Status

Fixed

[N6] [Suggestion] Missing zero address check

Category: Others

Content

1.In the ApprovedTokenManager contract, the owner can add any tokens to the whitelist, but there is a lack of zero address check here, which may affect users' normal transactions if the owner accidentally approves a zero address.

Code location:

swap-core/contracts/ApprovedTokenManager.sol#L13-16

```
function approveToken(address token, bool approved)  public onlyOwner{
  emit ApproveToken(token, approved);
  approvedToken[token]= approved;
}
```

2.Lack of zero address check during initialization, if the parameter is accidentally set to zero address, it may affect the normal function.

Code location:

light-dao/contracts/feeDistributor/feeDistributor.sol#L81-99

```
function initialize(
    address _votingEscrow,
    uint256 _startTime,
    address _token,
    address _stHOPE,
    address _emergencyReturn
) external initializer {
    __Ownable2Step_init();

    uint256 t = LibTime.timesRoundedByWeek(_startTime);
    startTime = t;
    lastTokenTime = t;
    timeCursor = t;
    emergencyReturn = _emergencyReturn;

    token = _token;
    stHOPE = _stHOPE;
```

```
        votingEscrow = _votingEscrow;
    }
```

light-dao/contracts/feeDistributor/GombocFeeDistributor.sol#L79-97

```
    function initialize(
        address _gombocController,
        uint256 _startTime,
        address _token,
        address _stHOPE,
        address _emergencyReturn
    ) external initializer {
        __Ownable2Step_init();

        uint256 t = LibTime.timesRoundedByWeek(_startTime);
        startTime = t;
        lastTokenTime = t;

        token = _token;
        stHOPE = _stHOPE;
        timeCursor = t;
        gombocController = _gombocController;
        emergencyReturn = _emergencyReturn;
    }
```

light-dao/contracts/feeDistributor/SwapFeeToVault.sol#L23-26

```
    constructor(BurnerManager _burnerManager, address _underlyingBurner) {
        burnerManager = _burnerManager;
        underlyingBurner = _underlyingBurner;
    }
```

light-dao/contracts/feeDistributor/UnderlyingBurner.sol#L50-65

```
    function initialize(
        address _hopeToken,
        address _feeDistributor,
        address _gombocFeeDistributor,
        address _emergencyReturn
    ) external initializer {
        __Ownable2Step_init();

        hopeToken = _hopeToken;
        feeDistributor = _feeDistributor;
```

```
        gombocFeeDistributor = _gombocFeeDistributor;
        emergencyReturn = _emergencyReturn;

        IERC20Upgradeable(hopeToken).approve(feeDistributor, 2 ** 256 - 1);
        IERC20Upgradeable(hopeToken).approve(gombocFeeDistributor, 2 ** 256 - 1);
    }
```

Solution

It is recommended to add a zero address check.

Status

Fixed

[N7] [Suggestion] Redundant code

Category: Others

Content

Redundant code for calling the _getPointBalanceOf function.

Code location:

light-dao/contracts/feeDistributor/FeeDistributor.sol#L202

```
    function _checkpointTotalSupply() internal {
        ...

        for (int i = 0; i < 20; i++) {
            ...
                _getPointBalanceOf(pt.bias, pt.slope, dt);
                veSupply[t] = _getPointBalanceOf(pt.bias, pt.slope, dt);
            }
            t += WEEK;
        }
        timeCursor = t;
    }
```

Solution

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

Status

Fixed

[N8] [Suggestion] Missing event records

Category: Others

Content

1.In the factory contract, the feeToSetter role can set the feeTo, feeToSetter and approvedTokenManager addresses, but there are no event logs.

Code location:

swap-core/contracts/UniswapV2Factory.sol#L47-60

```
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }

    function setApprovedTokenManager(IApprovedTokenManager _approvedTokenManager)
external {
        require(msg.sender == feeToSetter, 'UniswapV2: FORBIDDEN');
        approvedTokenManager = _approvedTokenManager;
    }
```

2.In the FeeDistributor, GombocFeeDistributor and UnderlyingBurner contract, the owner can set the emergencyReturn address to receive the withdrawn tokens, but there are no event logs.

Code location:

light-dao/contracts/feeDistributor/feeDistributor.sol#L458-460

```
    function setEmergencyReturn(address _addr) external onlyOwner {
        emergencyReturn = _addr;
    }
```

light-dao/contracts/feeDistributor/GombocFeeDistributor.sol#L468-470

```
    function setEmergencyReturn(address _addr) external onlyOwner {
        emergencyReturn = _addr;
```

```
    }
```

light-dao/contracts/feeDistributor/UnderlyingBurner.sol#L134-136

```
    function setEmergencyReturn(address _addr) external onlyOwner {
        emergencyReturn = _addr;
    }
```

3.In the LightSwapBurner and UnderlyingBurner contract, the owner can set arbitrary exchange routers.

Code location:

light-dao/contracts/feeDistributor/LightSwapBurner.sol#L33-35

```
    function setRouters(ISwapRouter[] calldata _routers) external onlyOwner {
        routers = _routers;
    }
```

light-dao/contracts/feeDistributor/UnderlyingBurner.sol#L100-102

```
    function setRouters(ISwapRouter[] calldata _routers) external onlyOwner {
        routers = _routers;
    }
```

Solution

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

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002303010001	SlowMist Security Team	2023.02.20 - 2023.03.01	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, 4 low risk, 3 suggestion vulnerabilities. All findings were fixed.

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