Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.04.26, the SlowMist security team received the StarryNift team's security audit application for StarryNift Contracts, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This audit focuses on the STAKE, TOKEN, and VESTING modules of the Starry project. Users can deposit specified tokens in LinearPool and get rewards, StarryNiftAirdrop contract is used to receive token airdrops, and Vesting contract is used to allow owners to create schedules and pay bonuses.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Lack of update of	Design Logic Audit	Critical	Fixed

NO	Title	Category	Level	Status
	user's data			
N2	Lack of checking for new start times	Design Logic Audit	Low	Fixed
N3	Missing event record	Others	Suggestion	Fixed
N4	Missing return value check	Others	Suggestion	Fixed
N5	Missing array length check	Design Logic Audit	Low	Fixed
N6	Missing zero address check	Others	Suggestion	Fixed
N7	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N8	Missing parameter checks when creating and modifying schedules	Design Logic Audit	Medium	Fixed

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/StarryNift/starry-token-contract

commit: 62e682eadacb99145821b576d8d34a77a0fc9d15

Fixed Version:

https://github.com/StarryNift/starry-token-contract

commit: bf51eaf893ce6bd8dfd80c87768b1e5d048ef9aa

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:

LinearPool			
Function Name	Visibility	Mutability	Modifiers
__LinearPool_init	Public	Can Modify State	initializer
pauseContract	External	Can Modify State	onlyOwner
unpauseContract	External	Can Modify State	onlyOwner
linearAdminRecoverFund	External	Can Modify State	onlyOwner
linearPoolLength	External	-	-
linearTotalStaked	External	-	linearValidatePoolById
linearAddPool	External	Can Modify State	onlyOwner
linearSetPool	External	Can Modify State	onlyOwner linearValidatePoolById
linearSetRewardDistributor	External	Can Modify State	onlyOwner
linearDeposit	External	Can Modify State	nonReentrant whenNotPaused linearValidatePoolById
linearWithdrawAll	External	Can Modify State	nonReentrant whenNotPaused linearValidatePoolById
linearClaimReward	External	Can Modify State	nonReentrant whenNotPaused linearValidatePoolById
linearDepositReward	External	Can Modify State	nonReentrant whenNotPaused linearValidatePoolById
linearPendingReward	Public	-	linearValidatePoolById
linearBalanceOf	External	-	linearValidatePoolById
linearUserStakingData	External	-	linearValidatePoolById
linearUserTotalStaked	External	-	-

LinearPool			
_linearDeposit	Internal	Can Modify State	-
_linearHarvest	Private	Can Modify State	-

StarryNiftAirdrop			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
getTokenAddress	Public	-	-
setTokenAddress	Public	Can Modify State	onlyOwner
getChainID	External	-	-
_hashClaimCallData	Internal	-	-
_verifyClaimCallData	Internal	-	-
claim	External	Can Modify State	whenNotPaused
setSigner	Public	Can Modify State	onlyOwner
getSigner	External	-	-

StarryNiftToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC20Permit ERC20Capped
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner
mint	External	Can Modify State	onlyOwner whenNotPaused
_beforeTokenTransfer	Internal	Can Modify State	whenNotPaused
_mint	Internal	Can Modify State	-

Vesting			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
createVestingSchedule	Public	Can Modify State	onlyOwner
getBeneficiary	Public	-	-
updateReleasedAmounts	Public	Can Modify State	onlyOwner
updateVestingSchedule	Public	Can Modify State	onlyOwner
_calcClaimableTimes	Internal	-	-
releasable	Public	-	-
release	Public	Can Modify State	onlyOwner
interval	Public	-	-
start	Public	-	-
end	Public	-	-
quickConfig	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Critical] Lack of update of user's data

Category: Design Logic Audit

Content

In the LinearPool contract, the user can deposit the reward into the pool again by calling the linearDepositReward function. However the values of stakingData.joinTime and pool.totalStaked are not updated when the deposit reward is done in this function.

This may result in unintended errors in withdrawal and reward calculations. (The number of total pool stakes does not match the number of user deposits).

Code Location:

contracts/stake/LinearPool.sol#L405-431

```
        function linearDepositReward(uint256 _poolId)
        external
        nonReentrant
        whenNotPaused
        linearValidatePoolById(_poolId)
        {
                address account = msg.sender;
                LinearStakingData storage stakingData = linearStakingData[_poolId][
                                        account
                        ];
                LinearPoolInfo storage pool = linearPoolInfo[_poolId];

                require(
                        block.timestamp >= stakingData.joinTime + pool.lockDuration,
                        "LinearStakingPool: still locked"
                );

                _linearHarvest(_poolId, account);

                if (stakingData.reward > 0) {
                        uint128 reward = stakingData.reward;
                        stakingData.reward = 0;
                        stakingData.balance = stakingData.balance + reward;
                        stakingData.updatedTime = block.timestamp.toUint128();
                        emit LinearDeposit(_poolId, account, reward);
                }
        }
```

Solution

It is recommended that the total pool stakes and the timing of user deposits be updated in a timely manner in the linearDepositReward function.

Status

Fixed

[N2] [Low] Lack of checking for new start times

Category: Design Logic Audit

Content

In the LinearPool contract, the owner role can modify the pool's configuration, including the pool's deposit start time and end time, by calling the linearSetPool function. However here only the new end time of the pool is checked against the old start time and mssing check the new start time of the pool. This may result in an error if the pool new start time is incorrectly set to exceed the pool new end time.

Code Location:

contracts/stake/LinearPool.sol#L253-273

```
    function linearSetPool(
            uint128 _poolId,
            uint128 _cap,
            uint64 _APR,
      uint128 _startJoinTime,
            uint128 _endJoinTime
    ) external onlyOwner linearValidatePoolById(_poolId) {
            LinearPoolInfo storage pool = linearPoolInfo[_poolId];

            require(
                    _endJoinTime >= block.timestamp &&
                    _endJoinTime > pool.startJoinTime,
                    "LinearStakingPool: invalid end join time"
            );

            linearPoolInfo[_poolId].cap = _cap;
            linearPoolInfo[_poolId].APR = _APR;
            linearPoolInfo[_poolId].startJoinTime = _startJoinTime;
            linearPoolInfo[_poolId].endJoinTime = _endJoinTime;
            emit LinearUpdatePool(_poolId, _APR, _cap, _endJoinTime);
    }
```

Solution

It is recommend that a check on the time of new deposits to the pool should be added here.

Status

Fixed

[N3] [Suggestion] Missing event record

Category: Others

Content

The following functions in several contracts are for event logging of key parameter settings.

Code Location:

contracts/token/StarryNiftAirdrop.sol

```
        function setTokenAddress(address _tokenAddress) public onlyOwner {
                tokenAddress = _tokenAddress;
        }

      ...

        function setSigner(address _signer) public onlyOwner {
                signer = _signer;
        }
```

contracts/vesting/Vesting.sol

```
        function createVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address tokenAddress,
                uint256 _interval,
                uint256 totalAmount,
                uint256[] memory _releaseAmounts,
                address[] memory beneficiary
        ) public onlyOwner {
                ...
        }

        ...

        function updateReleasedAmounts(uint256 scheduleId, uint256[] memory
_releaseAmounts) public onlyOwner {
                ...

        function updateVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address _tokenAddress,
                uint256 _interval,
                uint256 _releasedAmount,
                uint256 _totalAmount,
                address[] memory _beneficiary
        ) public onlyOwner {
```

```
        ...
    }
```

Solution

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

Status

Fixed

[N4] [Suggestion] Missing return value check

Category: Others

Content

1.In the StarryNiftAirdrop contract, the user can call the claim function to transfer tokens in the contract. But it does not check the return value. If external tokens do not adopt the EIP20 standard, it may lead to “false top-up” issues.

Code Location:

contracts/token/StarryNiftAirdrop.sol#L72

```
    function claim(ClaimCallData calldata data) external whenNotPaused {
        ...

        IERC20(tokenAddress).transfer(msg.sender, data.amount);
    }
```

2.In the Vesting contract, the owner can call the release function to transfer tokens in the contract. But it does not check the return value. If external tokens do not adopt the EIP20 standard, it may lead to “false top-up” issues.

Code Location:

contracts/vesting/Vesting.sol#L112

```
    function release(uint256 scheduleId) public onlyOwner {
        ...
```

```
                for (uint256 i = 0; i < schedule.beneficiary.length; i++) {
                        uint256 amount = releaseAmounts[schedule.beneficiary[i]] *
claimableTimes;
                        releasedAmount += amount;

IERC20(schedule.tokenAddress).transfer(schedule.beneficiary[i], amount);
                }

                ...
        }
```

Solution

It is recommended to add a check of the return value or use SafeERC20 library.

Status

Fixed

[N5] [Low] Missing array length check

Category: Design Logic Audit

Content

In the Vesting contract, the owner can create a new schedule by calling the createVestingSchedule function and can modify the released amounts by calling the updateReleasedAmounts function. However, there is no check on the length of the incoming array in either function, which can lead to unintended errors if the incoming array is too long.

Code Location:

contracts/vesting/Vesting.sol

```
        function createVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address tokenAddress,
                uint256 _interval,
                uint256 totalAmount,
                uint256[] memory _releaseAmounts,
                address[] memory beneficiary
        ) public onlyOwner {
                schedules[scheduleId] = VestingSchedule(_start, _end, tokenAddress,
```

```
_interval, 0, 0, totalAmount, beneficiary);
                for (uint256 index = 0; index < _releaseAmounts.length; index++) {
                        releaseAmounts[beneficiary[index]] = _releaseAmounts[index];
                }
        }

        ...

        function updateReleasedAmounts(uint256 scheduleId, uint256[] memory
_releaseAmounts) public onlyOwner {
                VestingSchedule memory schedule = schedules[scheduleId];
                for (uint256 index = 0; index < _releaseAmounts.length; index++) {
                        releaseAmounts[schedule.beneficiary[index]] =
_releaseAmounts[index];
                }
        }
```

Solution

It is recommended to add a check in the function that the length of the incoming releaseAmounts array and the beneficiary array should be equal.

Status

Fixed

[N6] [Suggestion] Missing zero address check

Category: Others

Content

Several of the following functions do not check for a zero address when setting a token address.

Code Location:

contracts/token/StarryNiftAirdrop.sol

```
        function setTokenAddress(address _tokenAddress) public onlyOwner {
                tokenAddress = _tokenAddress;
        }
```

Code Location:

contracts/vesting/Vesting.sol

```
        function createVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address tokenAddress,
                uint256 _interval,
                uint256 totalAmount,
                uint256[] memory _releaseAmounts,
                address[] memory beneficiary
        ) public onlyOwner {
                schedules[scheduleId] = VestingSchedule(_start, _end, tokenAddress,
_interval, 0, 0, totalAmount, beneficiary);
                for (uint256 index = 0; index < _releaseAmounts.length; index++) {
                        releaseAmounts[beneficiary[index]] = _releaseAmounts[index];
                }
        }

        ...

        function updateVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address _tokenAddress,
                uint256 _interval,
                uint256 _releasedAmount,
                uint256 _totalAmount,
                address[] memory _beneficiary
        ) public onlyOwner {
                schedules[scheduleId].start = _start;
                schedules[scheduleId].end = _end;
                schedules[scheduleId].tokenAddress = _tokenAddress;
                schedules[scheduleId].interval = _interval;
                schedules[scheduleId].releasedAmount = _releasedAmount;
                schedules[scheduleId].totalAmount = _totalAmount;
                schedules[scheduleId].beneficiary = _beneficiary;
        }
```

Solution

It is recommended that an address non-zero check should be added.

Status

Fixed

[N7] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the LinearPool contract, the owner role can set the owner role can transfer any tokens in the contract by calling the linearAdminRecoverFund function. In addition to this, the owner role can set the sender of the reward tokens by calling the linearSetRewardDistributor function. If the privilege is lost or misused, there may be an impact on the user's funds.

Code Location:

contracts/stake/LinearPool.sol

```
function linearAdminRecoverFund(
        address _token,
        address _to,
        uint256 _amount
) external onlyOwner {
        require(
                IERC20(_token).balanceOf(address(this)) >= _amount,
                "LinearStakingPool: not enough balance"
        );
        IERC20(_token).safeTransfer(_to, _amount);
        emit AdminRecoverFund(_token, _to, _amount);
}

...

function linearSetRewardDistributor(address _linearRewardDistributor)
external
onlyOwner
{
        require(
                _linearRewardDistributor != address(0),
                "LinearStakingPool: invalid reward distributor"
        );

        emit ChangeRewardDistributor(
                linearRewardDistributor,
                _linearRewardDistributor
        );
        linearRewardDistributor = _linearRewardDistributor;
}
```

Solution

It is recommended that in the early stages of the project, the core role like the owner should use multi-signatures and the time-lock contract to avoid single-point risks. After the project is running stably, the authority of the core role should be handed over to community governance for management.

Status

Acknowledged; The project team responded that they will move the owner's permissions to a multi-signature address in the future.

[N8] [Medium] Missing parameter checks when creating and modifying schedules

Category: Design Logic Audit

Content

1.In the Vesting contract, the owner role passes in a totalAmount parameter when creating and modifying timesheets, and this parameter is not involved in the contract's checking when setting release quantities. There are no checks or warnings if the sum of the number of tokens acquired by each beneficiary exceeds the totalAmount set, or if the number of released tokens set exceeds the totalAmount.

Code Location:

contracts/vesting/Vesting.sol

```
        function createVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address tokenAddress,
                uint256 _interval,
                uint256 totalAmount,
                uint256[] memory _releaseAmounts,
                address[] memory beneficiary
        ) public onlyOwner {
                schedules[scheduleId] = VestingSchedule(_start, _end, tokenAddress,
_interval, 0, 0, totalAmount, beneficiary);
                for (uint256 index = 0; index < _releaseAmounts.length; index++) {
                        releaseAmounts[beneficiary[index]] = _releaseAmounts[index];
                }
        }

        function updateReleasedAmounts(uint256 scheduleId, uint256[] memory
```

```
_releaseAmounts) public onlyOwner {
                VestingSchedule memory schedule = schedules[scheduleId];
                for (uint256 index = 0; index < _releaseAmounts.length; index++) {
                        releaseAmounts[schedule.beneficiary[index]] =
_releaseAmounts[index];
                }
        }

        function updateVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address _tokenAddress,
                uint256 _interval,
                uint256 _releasedAmount,
                uint256 _totalAmount,
                address[] memory _beneficiary
        ) public onlyOwner {
                schedules[scheduleId].start = _start;
                schedules[scheduleId].end = _end;
                schedules[scheduleId].tokenAddress = _tokenAddress;
                schedules[scheduleId].interval = _interval;
                schedules[scheduleId].releasedAmount = _releasedAmount;
                schedules[scheduleId].totalAmount = _totalAmount;
                schedules[scheduleId].beneficiary = _beneficiary;
        }
```

2.In the Vesting contract, the owner role can set the release start time, end time, and interval. However, these incoming parameters are not checked when creating or modifying a timetable, which can cause unintended errors if the end time is greater than or equal to the start time, or if the interval exceeds the difference between the end time and the start time.

Code Location:

contracts/vesting/Vesting.sol

```
        function createVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address tokenAddress,
                uint256 _interval,
                uint256 totalAmount,
                uint256[] memory _releaseAmounts,
                address[] memory beneficiary
```

```
        ) public onlyOwner {
                schedules[scheduleId] = VestingSchedule(_start, _end, tokenAddress,
_interval, 0, 0, totalAmount, beneficiary);
                for (uint256 index = 0; index < _releaseAmounts.length; index++) {
                        releaseAmounts[beneficiary[index]] = _releaseAmounts[index];
                }
        }

        function updateVestingSchedule(
                uint256 scheduleId,
                uint256 _start,
                uint256 _end,
                address _tokenAddress,
                uint256 _interval,
                uint256 _releasedAmount,
                uint256 _totalAmount,
                address[] memory _beneficiary
        ) public onlyOwner {
                schedules[scheduleId].start = _start;
                schedules[scheduleId].end = _end;
                schedules[scheduleId].tokenAddress = _tokenAddress;
                schedules[scheduleId].interval = _interval;
                schedules[scheduleId].releasedAmount = _releasedAmount;
                schedules[scheduleId].totalAmount = _totalAmount;
                schedules[scheduleId].beneficiary = _beneficiary;
        }
```

Solution

It is recommended to check the incoming parameters when creating and modifying the configuration of the shedule.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002404290003	SlowMist Security Team	2024.04.26 - 2024.04.29	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 critical risk, 2 medium risk, 2 low risk, 3 suggestion vulnerabilities. 1 medium risk vulnerability was acknowledged and other findings were fixed. Since the code has not yet been deployed to the main network and the owner role's permission has not yet been transferred, the reported level is currently medium risk for the time being. In the future, we will reduce the level of risk after confirming the transfer of authority.

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