Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.04.06, the SlowMist security team received the team's security audit application for IoTeX - SystemStaking, 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

Staking contract that represents the staking bucket as a non-fungible token

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Incorrect variable used in Staked event emission	Design Logic Audit	Low	Fixed

NO	Title	Category	Level	Status
N2	Using the transfer function to transfer ETH may cause assets to be locked.	Unsafe External Call Audit	Medium	Fixed
N3	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N4	Default penalty rate can be dangerous	Others	Low	Fixed
N5	Note on implementation of _isActiveBucketType	Design Logic Audit	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Codebase:

https://github.com/iotexproject/iip13-contracts/blob/main/src/SystemStaking.sol

commit: 8ce70c160d4f8ee4c3c301a11c718faabd08949a

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:

SystemStaking			
Function Name	Visibility	Mutability	Modifiers
pause	external	Can Modify State	onlyOwner
unpause	external	Can Modify State	onlyOwner

SystemStaking			
withdrawFee	external	Can Modify State	onlyOwner
setEmergencyWithdrawPenaltyRate	external	Can Modify State	onlyOwner
emergencyWithdrawPenaltyRate	external	view	-
accumulatedWithdrawFee	external	view	-
addBucketType	external	Can Modify State	onlyOwner
deactivateBucketType	external	Can Modify State	onlyOwner
activateBucketType	external	Can Modify State	onlyOwner
isActiveBucketType	external	view	-
numOfBucketTypes	public	view	-
bucketTypes	external	view	-
blocksToUnstake	external	view	-
blocksToWithdraw	public	view	-
bucketOf	external	view	-
stake	external	payable	whenNotPaused
stake	external	payable	whenNotPaused
stake	external	payable	whenNotPaused
unlock	external	Can Modify State	whenNotPaused; onlyTokenOwner
unlock	external	Can Modify State	whenNotPaused
lock	external	Can Modify State	whenNotPaused; onlyTokenOwner
lock	external	Can Modify State	whenNotPaused

SystemStaking			
unstake	external	Can Modify State	whenNotPaused; onlyTokenOwner
unstake	external	Can Modify State	whenNotPaused
withdraw	external	Can Modify State	whenNotPaused; onlyTokenOwner
withdraw	external	Can Modify State	whenNotPaused
emergencyWithdraw	external	Can Modify State	onlyTokenOwner
merge	external	payable	whenNotPaused
extendDuration	external	Can Modify State	whenNotPaused; onlyTokenOwner
increaseAmount	external	payable	whenNotPaused; onlyTokenOwner
changeDelegate	external	Can Modify State	whenNotPaused; onlyTokenOwner
changeDelegates	external	Can Modify State	whenNotPaused
lockedVotesTo	external	view	-
unlockedVotesTo	external	view	-

4.3 Vulnerability Summary

[N1] [Low] Incorrect variable used in Staked event emission

Category: Design Logic Audit

Content

In the emit statement of the Staked event, it is observed that the msgValue variable is used to emit the amount of the stake, which is not correct. The correct variable to use is _amount.

```
function stake(
    uint256 _amount,
    uint256 _duration,
        bytes12 _delegate,
        uint256 _count
    ) external payable whenNotPaused returns (uint256 firstTokenId_) {
        uint256 msgValue = msg.value;
        require(_count > 0 && _amount * _count == msgValue, "invalid parameters");
        uint256 index = _bucketTypeIndex(_amount, _duration);
        _assertOnlyActiveBucketType(index);
        unchecked {
            firstTokenId_ = __currTokenId + 1;
        }
        for (uint256 i = 0; i < _count; i = unsafeInc(i)) {
            _stake(index, _delegate);
            emit Staked(firstTokenId_ + i, _delegate, msgValue, _duration);
//SlowMist// msgValue should be _amount
        }

        return firstTokenId_;
    }
```

Solution

We suggest updating the code to use _amount instead of msgValue.

Status

Fixed

[N2] [Medium] Using the transfer function to transfer ETH may cause assets to be locked.

Category: Unsafe External Call Audit

Content

`transfer` use a fixed gas limit (2300). Plus, User can not transfer there bucket token once they unstake. If the recipient is a contract and implement a `fallback` function which used more than 2300 gas, then user can suffer from lock of fund.

src/SystemStaking.sol

```
93:         _recipient.transfer(_amount);
593:        _recipient.transfer(amount - fee_);
```

User can not transfer there bucket token once they unstake.

```
    function _beforeTokenTransfer(
        address _from,
        address _to,
        uint256 _firstTokenId,
        uint256 _batchSize
    ) internal override {
        require(_batchSize == 1, "batch transfer is not supported");
        require(
            _to == address(0) || !_isTriggered(__buckets[_firstTokenId].unstakedAt),
            "cannot transfer unstaked token"
        );
        super._beforeTokenTransfer(_from, _to, _firstTokenId, _batchSize);
    }
```

Solution

use `call` to transfer native token instead

Status

Fixed

[N3] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

Owner can perform critical functions such as `pause`, `withdrawFee`, `setEmergencyWithdrawPenaltyRate`.

This lead to the risk of excessive permissions of the `owner` role in this two scenario.

1. `owner` can perform `pause`, and user can suffer from lock of fund
2. `owner` can frontrun user's `emergencyWithdraw` and `setEmergencyWithdrawPenaltyRate` to 100

   which can drain out user's fund.

Code location:

src/SystemStaking.sol

```
function pause() external onlyOwner {
    _pause();
}

function withdrawFee(uint256 _amount, address payable _recipient) external onlyOwner
{
    __accumulatedWithdrawFee -= _amount;
    _recipient.transfer(_amount);
    emit FeeWithdrawal(_recipient, _amount);
}

function setEmergencyWithdrawPenaltyRate(uint256 _rate) external onlyOwner {
    _setEmergencyWithdrawPenaltyRate(_rate);
}
```

Solution

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

Status

Acknowledged; Ownership will be hold in a multisig wallet and timelock will be implemented later.

[N4] [Low] Default penalty rate can be dangerous

Category: Others

Content

We set penalty rate to 100% in `constructor`. This can confiscate all funds of the user as a penalty.

```
    constructor() ERC721("BucketNFT", "BKT") {
        _setEmergencyWithdrawPenaltyRate(100);
    }
```

Solution

Either set a less strict default or document this variable explicitly.

Status

Fixed; Fixed by remove the emergency withdraw logic.

[N5] [Suggestion] Note on implementation of _isActiveBucketType

Category: Design Logic Audit

Content

When passing a non-exist index of bucket type to `_isActiveBucketType`, the return value will be `true`.

Note that the current implementation `_isActiveBucketType` is not a problem due to `_isActiveBucketType` always get index from `_bucketTypeIndex` which can not return invalid index (index of non-exist bucket type).

By fixing it we can:

1. Avoid potential problem in future development.
2. Convey the clearer logic required by the specification.

```
function _isActiveBucketType(uint256 _index) internal view returns (bool) {
    return __bucketTypes[_index].activatedAt <= block.number;
}
```

Solution

use `return __bucketTypes[_index].activatedAt <= block.number && __bucketTypes[index].activatedAt != 0;` instead.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002304110001	SlowMist Security Team	2023.04.06 - 2023.04.11	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 medium risk, 2 low risk, 1 suggestion.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.

SLOWMIST

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist