Smart Contract Security Audit Report

Contents

1. Executive Summary

On Nov. 30, 2020, the SlowMist security team received the Tom Finance team's security audit application for Tom Finance, 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.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module 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.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and 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:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack

- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

TOM Finance is launching an automated market making (AMM) decentralized exchange (DEX) that is built on the Ethereum blockchain with the purpose of enhancing liquidity for ERC-20 gold stable coins on decentralized exchanges.

Audit version file information

audit files:

https://github.com/TomFinance/TOM_Contracts

commit: 8badddd7ef0b1b268802cabb2a44ff46829cacd2

3.2 Project Structure

```
├── Pool.sol
├── README.md
└── Token.sol
```

4. Code Overview

4.1 Main Contract address

Contract Name	Contract Address
TMTG-LBXC (Pool)	0x2A70605e53a2a596E04df8A775E0e8C9fEd62F9a
TOM (ERC20 Token)	0xF7970499814654CD13Cb7B6E7634A12a7A8A9ABc

4.2 Contracts Description

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

SafeMath			
Function Name	Visibility	Mutability	Modifiers
add	Internal	-	-
sub	Internal	-	-
sub	Internal	-	-
mul	Internal	-	-
div	Internal	-	-
div	Internal	-	-
mod	Internal	-	-
mod	Internal	-	-

ERC20			
Function Name	Visibility	Mutability	Modifiers
name	Public	-	-
symbol	Public	-	-
decimals	Public	-	-
totalSupply	Public	-	-
balanceOf	Public	-	-
transfer	Public	Can modify state	-

allowance	Public	-	-
approve	Public	Can modify state	-
transferFrom	Public	Can modify state	-
increaseAllowance	Public	Can modify state	-
decreaseAllowance	Public	Can modify state	-
_transfer	Internal	Can modify state	-
_mint	Internal	Can modify state	-
_burn	Internal	Can modify state	-
_approve	Internal	Can modify state	-
_burnFrom	Internal	Can modify state	-

Pool			
Function Name	Visibility	Mutability	Modifiers
claimAllReward	External	Can modify state	-
stake	External	Can modify state	-
claimAndUnstake	External	Can modify state	-
unstakeAll	External	Can modify state	-
emergencyExit	External	Can modify state	-
inquiryDeposit	External	-	-
inquiryRemainReward	External	-	-
inquiryExpectedReward	External	-	-
_registAddress	Internal	Can modify state	-
_withdraw	Internal	Can modify state	-
_updateAllReward	Internal	Can modify state	-
_updateReward	Internal	Can modify state	-
_elapsedBlock	Internal	-	-
_calculateEarn	Internal	-	-
changeRewardRate	External	Can modify state	-
_max	Private	-	-
_min	Private	-	-

4.3 Code Audit

4.3.1 Medium-risk vulnerabilities

4.3.1.1 Risk of excessive authority

The Token contract will mint all reward tokens to the owner role at one time during deployment. If the owner role does not distribute all these tokens to the Pool contract, this will lead to the risk of excessive owner role authority.

It is suggested to set the minter role for the Token contract and set the Pool contract address to minter. When the Pool contract needs to distribute rewards, the Pool contract mints tokens to users in the Token contract.

It should be noted that only the owner of the minter role can be added, and the owner role should be set to a timelock contract (such as Compound's timelock contract), or use community governance to control the owner role to avoid the risk of excessive owner role authority.

Code location: Token.sol file, line 110

```
    constructor (
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 totalSupply
    ) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
        _mint(msg.sender, totalSupply);
    }
```

Fix status: The total amount of tokens in the Tom contract is 25,000 (decimals is 18), and the project party has transferred all tokens to the Pool contract.

The reference transaction hash:

0xf4fe8fbda1c1bb60d7ee0def7ba5c4ea91a1d10fdc69561296e8fcb9b05ae006

4.3.2 Low-risk vulnerabilities

4.3.2.1 Risk of excessive authority

The CONSTRUCTOR_ADDRESS role can arbitrarily modify the *rewardPerBlock* parameter through the changeRewardRate function. The *rewardPerBlock* parameter affects the user's income, which will lead to the risk of excessive permissions in the CONSTRUCTOR_ADDRESS role.

It is suggested to set the CONSTRUCTOR_ADDRESS role to the timelock contract (such as Compound's timelock contract), or set a variable range for the *rewardPerBlock* parameter.

Code location: Pool.sol file, line 191

```
function changeRewardRate (uint256 rate) external {
        require(CONSTRUCTOR_ADDRESS == msg.sender, "Only constructor can do this");
        // _updateAllReward();
        rewardPerBlock = rate;
    }
```

Fix status: The project party has not fixed this issues yet.

4.3.3 Enhancement Suggestions

4.3.3.1 Part of the code is redundant

The visibility of the _burn function and _burnFrom function is internal, and users cannot directly call these functions.

It is suggested to delete these redundant functions.

Code location: Token.sol file, lines 180, 196

```
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "ERC20: burn amount exceeds allowance"));
    }
```

The visibility of the _updateAllReward function is internal, but no other function calls the _updateAllReward function.

It is suggested to delete these redundant functions.

Code location: Pool.sol file, line 164

```
function _updateAllReward () internal {
        for(uint256 i=0; i<PARTICIPANT_LIST.length; i++){
            _updateReward(PARTICIPANT_LIST[i]);
        }
    }
```

Fix status: The project party has not fixed this issues yet.

4.3.3.2 Risk of False top-up

If the transferFrom function of the stakeToken contract uses a False top-up method (such as HT token), it will lead to fake stake risks.

For example, when the user calls the stake function, the contract will call the transferFrom function of the stakeToken contract. Under normal circumstances, if the user's balance is insufficient, the

transaction will be rolled back directly, and the user cannot successfully perform the stake operation.

But if the transferFrom function of the stakeToken contract uses a False top-up method, the transferFrom function will return false if the user's balance is insufficient, and the entire transaction will not be rolled back, so the contract will continue to execute:

STAKED_AMOUNT[msg.sender] = STAKED_AMOUNT[msg.sender].add(amount);

This leads to the fact that the user's token is not actually transferred to the Pool contract, but the Pool contract records the user's successful stake.

It is suggested to check the contract balance before and after the stake.

Code location: Pool.sol file, line 111

```
function stake (uint256 amount) external {
        _registAddress(msg.sender);
        _updateReward(msg.sender);
        stakeToken.transferFrom(msg.sender, address(this), amount);
        STAKED_AMOUNT[msg.sender] = STAKED_AMOUNT[msg.sender].add(amount);
        totalStaked = totalStaked.add(amount);
    }
```

Fix status: The project party has not fixed this issues yet.

4.3.3.3 Income change issue

Before the CONSTRUCTOR_ADDRESS role modifies the rewardPerBlock parameter through the changeRewardRate function, the stake user's income is not first settled. This will cause the user's previous earnings to change when the rewardPerBlock parameter is changed.

It is suggested to settle the user's income before changing the rewardPerBlock parameter.

Code location: Pool.sol file, line 187, 194

```
function _calculateEarn (uint256 elapsed, uint256 staked) internal view returns (uint256) {
        if(staked == 0){return 0;}
```

```
        return elapsed.mul(staked).mul(rewardPerBlock).div(totalStaked);
    }

.....

    function changeRewardRate (uint256 rate) external {
        require(CONSTRUCTOR_ADDRESS == msg.sender, "Only constructor can do this");
        // _updateAllReward();
        rewardPerBlock = rate;
    }
```

Fix status: The project party has not fixed this issues yet.

5. Audit Result

5.1 Conclusion

Audit Result : Low Risks

Audit Number : 0X002012030002

Audit Date : Dec. 03 2020

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist Team's analysis tool audit of the codes for security issues. There are seven issues five during the audit. There are one medium-risk vulnerabilities and one low-risk vulnerabilities. We also provide three enhancement suggestions. The project party has only fixed the medium-risk vulnerabilities. At present, the Tom Finance project still has the risk of excessive owner 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 base 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 this 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