Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.02.23, the SlowMist security team received the team's security audit application for TiTi 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.

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.

Level	Description
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:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Auidt Scope: Does not include staking and use2earn sections

Audit Version:

https://github.com/TiTi-Finance/TiTi-Core-Protocol

commit: 418189b4665a1ba07e00d0412862ab8a696146e8

Fixed Version:

https://github.com/TiTi-Finance/TiTi-Core-Protocol

commit: a41dd015e843c6d0076e4c74d9bd9878f741a7e1

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	Medium	Confirmed
N2	kLast update issue	Design Logic Audit	Medium	Fixed

4 Code Overview

4.1 Contracts Description

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:

TiUSDToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC20Permit
snapshot	External	Can Modify State	onlyRole
mint	External	Can Modify State	onlyRole
_beforeTokenTransfer	Internal	Can Modify State	-
reorders	External	Can Modify State	onlyRole
setNewAdmin	External	Can Modify State	onlyRole

TiTiToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20 ERC20Permit
snapshot	External	Can Modify State	onlyRole
mint	External	Can Modify State	onlyRole
_beforeTokenTransfer	Internal	Can Modify State	-
_afterTokenTransfer	Internal	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
setNewAdmin	External	Can Modify State	onlyRole
setNewMinters	External	Can Modify State	onlyRole

BaseVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
withdraw	External	Can Modify State	onlyOwner

ProtocolFeeVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseVault

RainyDayFundVault

RainyDayFundVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseVault

TiTiOracles			
Function Name	Visibility	Mutability	Modifiers
_updatePrice	Internal	Can Modify State	-
getTiUSDPrice	External	-	-

MarketMakerFund			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
addLiquidity	External	Can Modify State	onlyEOA nonReentrant whenNotPaused
removeLiquidity	External	Can Modify State	onlyEOA nonReentrant whenNotPaused
withdrawAll	External	Can Modify State	onlyEOA nonReentrant whenNotPaused
getShareValue	Public	-	-
getUserShareValue	External	-	-
setNewMAMM	External	Can Modify State	onlyOwner
setNewReordersController	External	Can Modify State	onlyOwner
setNewLPStakingPool	External	Can Modify State	onlyOwner

MarketMakerFund			
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner

IRewardDistributionRecipient			
Function Name	Visibility	Mutability	Modifiers
setRewardDistribution	External	Can Modify State	onlyOwner

LPTokenWrapper			
Function Name	Visibility	Mutability	Modifiers
totalSupply	Public	-	-
balanceOf	Public	-	-
stake	Public	Can Modify State	-
withdraw	Public	Can Modify State	-

MMFLPStakingPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
startNewEpoch	External	Can Modify State	onlyRewardDistribution updateReward
lastTimeRewardApplicable	Public	-	-
rewardPerToken	Public	-	-

MMFLPStakingPool			
earned	Public	-	-
stake	Public	Can Modify State	onlyMMF updateReward checkStart
withdraw	Public	Can Modify State	onlyMMF updateReward checkStart
getReward	External	Can Modify State	updateReward onlyMMF checkStart
stake	External	Can Modify State	updateReward checkStart
withdraw	External	Can Modify State	updateReward checkStart
getReward	External	Can Modify State	updateReward checkStart
governanceRecoverUnsupported	External	Can Modify State	onlyOwner

MAMMSwapPair			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getDepth	Public	-	-
getMMFFunds	Public	-	-
_update	Private	Can Modify State	-
mintFee	External	Can Modify State	nonReentrant onlyReordersController
addLiquidity	External	Can Modify State	nonReentrant onlyMMF

MAMMSwapPair			
removeLiquidity	External	Can Modify State	nonReentrant onlyMMF
_swap	Internal	Can Modify State	-
_getAmountOut	Internal	-	-
mint	External	Can Modify State	onlyEOA nonReentrant whenNotPaused
redeem	External	Can Modify State	onlyEOA nonReentrant whenNotPaused
sync	External	Can Modify State	nonReentrant whenNotPaused onlyReordersController
pavAllocation	External	Can Modify State	nonReentrant onlyReordersController whenNotPaused
migrate	External	Can Modify State	nonReentrant onlyOwner
setNewReordersController	External	Can Modify State	onlyOwner
setFeeTo	External	Can Modify State	onlyOwner
setNewMMF	External	Can Modify State	onlyOwner
setPeriod	External	Can Modify State	onlyOwner
setIsAllowedContractsCall	External	Can Modify State	onlyOwner
pause	External	Can Modify State	onlyOwner
unpause	External	Can Modify State	onlyOwner

TiTiGovernor

TiTiGovernor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Governor GovernorSettings GovernorVotes GovernorVotesQuorumFraction GovernorTimelockControl
votingDelay	Public	-	-
votingPeriod	Public	-	-
quorum	Public	-	-
getVotes	Public	-	-
state	Public	-	-
propose	Public	Can Modify State	-
proposalThreshold	Public	-	-
_execute	Internal	Can Modify State	-
_cancel	Internal	Can Modify State	-
_executor	Internal	-	-
supportsInterface	Public	-	-

TiTiTimelockController			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	TimelockController

ReOrdersController

ReOrdersController			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
reorders	External	Can Modify State	nonReentrant whenNotPaused
_reorders	Internal	Can Modify State	-
sync	External	Can Modify State	nonReentrant whenNotPaused onlyMMF
setNewMAMM	External	Can Modify State	onlyOwner
setNewMMF	External	Can Modify State	onlyOwner
setNewPriceDelta	External	Can Modify State	onlyOwner
setNewDuration	External	Can Modify State	onlyOwner
setNewAllocation	External	Can Modify State	onlyOwner
pause	External	Can Modify State	nonReentrant onlyOwner
unpause	External	Can Modify State	nonReentrant onlyOwner

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

- In the TiUSDToken contract, DEFAULT_ADMIN_ROLE can set any user to MINTER_ROLE through the grantRole function. MINTER_ROLE can mint TiUSD tokens arbitrarily through the mint function, which will lead to the risk of excessive DEFAULT_ADMIN_ROLE authority.
- The same is true for TiTiToken.

- In the MarketMakerFund contract, the owner can modify the parameters of mammSwapPair, reordersController and lpStakingPool through the functions setNewMAMM, setNewReordersController, and setNewLPStakingPool. This will lead to the risk of excessive owner permissions.
- In the MAMMSwapPair contract, the owner role can migrate the tokens in the pair to the specified address through the migrate function. And the owner role can modify the reordersController, feeTo and mmf parameters respectively through the setNewReordersController, setFeeTo, and setNewMMF functions. This will lead to the risk of excessive owner permissions.
- In the ReOrdersController contract, the owner can modify the sensitive parameters in the contract through the setNewMAMM, setNewMMF, setNewDuration and setNewAllocation functions, which will lead to the risk of excessive owner permissions.

Code location:

contracts/TiUSDToken.sol

```
    function mint(address to, uint256 amount) external onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }
```

contracts/TiTiToken.sol

```
    function mint(address to, uint256 amount) external onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }
```

contracts/mmf/MarketMakerFund.sol

```
    function setNewMAMM(IMAMMSwapPair _mammSwapPair) external onlyOwner {
        require(address(_mammSwapPair) != address(0), "MarketMakerFund: Cannot be
address(0)");
        address oldMAMM = address(mammSwapPair);
        mammSwapPair = _mammSwapPair;
        emit NewMAMM(oldMAMM, address(_mammSwapPair));
```

```
    }

    function setNewReordersController(IReOrdersController _reordersController)
external onlyOwner {
        require(address(_reordersController) != address(0), "MarketMakerFund: Cannot
be address(0)");
        address oldReorders = address(reordersController);
        reordersController = _reordersController;
        emit NewReordersController(oldReorders, address(reordersController));
    }

    function setNewLPStakingPool(IMMFLPStakingPool _lpStakingPool) external onlyOwner
{
        require(address(_lpStakingPool) != address(0), "MarketMakerFund: Cannot be
address(0)");
        address oldLPStakingPool = address(lpStakingPool);
        lpStakingPool = _lpStakingPool;
        _approve(address(this), address(lpStakingPool), type(uint256).max);
        emit NewLPStakingPool(oldLPStakingPool, address(_lpStakingPool));
    }
```

contracts/mamm/MAMMSwapPair.sol

```
    function migrate(address _to) external nonReentrant onlyOwner {
        IERC20 _token0 = token0;
        IERC20 _token1 = token1;

        (uint112 _fund0, uint112 _fund1,) = getDepth();

        uint balance0 = _token0.balanceOf(address(this));
        uint balance1 = _token1.balanceOf(address(this));

        _token0.safeTransfer(_to, balance0);
        _token1.safeTransfer(_to, balance1);

        // Clear all status
        _update(0, 0, _fund0, _fund1);
        mmfFund0 = uint(0);
        mmfFund1 = uint(0);

        emit Migrate(_to, fund0, fund1, mmfFund0, mmfFund1);
    }
```

```
    function setNewReordersController(address _reordersController) external onlyOwner
{
        address oldReorders = reordersController;
        reordersController = _reordersController;
        emit NewReordersController(oldReorders, _reordersController);
    }

    function setFeeTo(address _feeTo) external onlyOwner {
        address oldFeeTo = feeTo;
        feeTo = _feeTo;
        emit NewFeeTo(oldFeeTo, _feeTo);
    }

    function setNewMMF(address _mmf) external onlyOwner {
        require(_mmf != address(0), "MAMMSwapPair: Cannot be address(0)");
        address oldMMF = mmf;
        mmf = _mmf;
        emit NewMMF(oldMMF, _mmf);
    }
```

contracts/controller/ReOrdersController.sol

```
    function setNewMAMM(IMAMMSwapPair _mamm) external onlyOwner {
        require(address(_mamm) != address(0), "ReOrdersController: Cannot be
address(0)");
        address oldMAMM = address(mamm);
        mamm = _mamm;
        emit NewMAMM(oldMAMM, address(_mamm));
    }

   function setNewMMF(address _mmf) external onlyOwner {
        require(_mmf != address(0), "ReOrdersController: Cannot be address(0)");
        address oldMMF = mmf;
        mmf = _mmf;
        emit NewMMF(oldMMF, _mmf);
    }

   function setNewAllocation(
        uint256 _mmfRewardsAllocation,
        uint256 _rainyDayFundAllocation,
        uint256 _protocolFeeAllocation,
```

```
        address _rainyDayFundVault,
        address _protocolFeeVault
    ) external onlyOwner {
        require(_rainyDayFundVault != address(0), "ReOrdersController: Cannot be
address(0)");
        require(_protocolFeeVault != address(0), "ReOrdersController: Cannot be
address(0)");

        uint256 totalAllocation = _mmfRewardsAllocation + _rainyDayFundAllocation +
_protocolFeeAllocation;

        require(totalAllocation == 1e18, "ReOrdersController: totalAllocation must be
100%");

        mmfRewardsAllocation = _mmfRewardsAllocation;
        rainyDayFundAllocation = _rainyDayFundAllocation;
        protocolFeeAllocation = _protocolFeeAllocation;
        rainyDayFundVault = _rainyDayFundVault;
        protocolFeeVault = _protocolFeeVault;

        emit NewAllocation(
           _mmfRewardsAllocation,
           _rainyDayFundAllocation,
           _protocolFeeAllocation,
           _rainyDayFundVault,
           _protocolFeeVault
        );
    }
```

Solution

- It is recommended to transfer ownership of DEFAULT_ADMIN_ROLE to community governance.
- It is recommended to transfer ownership of DEFAULT_ADMIN_ROLE to community governance or set a minting cap for TiTi tokens.
- It is recommended to transfer the ownership of the MarketMakerFund contract to community governance.
- It is recommended to transfer the ownership of the MAMMSwapPair contract to community governance.
- It is recommended to transfer the ownership of the ReOrdersController contract to community governance.

Status

Confirmed; Since the protocol has not yet been deployed to the mainnet, and the ownership of the contract has not been transferred to community governance, there is still a risk of excessive authority.

[N2] [Medium] kLast update issue

Category: Design Logic Audit

Content

In the MAMMSwapPair contract, the addLiquidity function and the removeLiquidity function users add/remove liquidity. It will cause fund0&fund1 in the contract to change, but kLast is not updated accordingly. This will result in inaccurate kLast obtained during mintFee operation.

Code location: contracts/mamm/MAMMSwapPair.sol

```
    function addLiquidity() external nonReentrant onlyMMF {
        uint112 _fund0 = fund0;
        uint112 _fund1 = fund1;
        uint balance0 = token0.balanceOf(address(this));
        uint balance1 = token1.balanceOf(address(this));
        uint amount0 = balance0 - _fund0;
        uint amount1 = balance1 - _fund1;

        mmfFund0 += amount0;
        mmfFund1 += amount1;

        _update(balance0, balance1, _fund0, _fund1);

        emit AddLiquidity(amount0, amount1);
    }

    function removeLiquidity(uint _amount0, uint _amount1) external nonReentrant
onlyMMF {
        uint112 _fund0 = fund0;
        uint112 _fund1 = fund1;
        IERC20 _token0 = token0;
        IERC20 _token1 = token1;

        _token0.safeTransfer(mmf, _amount0);
```

```
        _token1.safeTransfer(mmf, _amount1);

        uint balance0 = _token0.balanceOf(address(this));
        uint balance1 = _token1.balanceOf(address(this));

        mmfFund0 = mmfFund0 - _amount0;
        mmfFund1 = mmfFund1 - _amount1;

        _update(balance0, balance1, _fund0, _fund1);

        emit RemoveLiquidity(_amount0, _amount1);
    }
```

Solution

It is recommended to perform mintFee operation and update kLast when add/remove liquidity.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002203040007	SlowMist Security Team	2022.02.23 - 2022.03.04	Medium 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 vulnerabilities. And 1 medium risk vulnerability was confirmed and being fixed; All other findings were fixed. Since the protocol has not yet been deployed to the mainnet, and the ownership of the contract has not been transferred to community governance, there is still a risk of excessive 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