Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.07.12, the SlowMist security team received the Oddz Finance team's security audit application for Oddz 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 party 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 Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit Version:

Part 1:

https://github.com/oddz-finance/oddz-contracts

commit: c4dc64f668d4d944ce87e6c35680a4eade79a45e

Part 2:

https://github.com/oddz-finance/oddz-contracts

commit: 8692693a7b4d9475a6d794b49285a66e9c19f916

Audit Scope:

contracts/Pool/OddzStrategyManager.sol

contracts/Pool/OddzWriteStrategy.sol

oddz-contracts-master-07.12.zip(SHA256):

044f56010e3b7c7a438620f54a4389741ee7cd2a041dc6b9e58a0079214fed66

Fixed Version：

https://github.com/oddz-finance/oddz-contracts

commit: 186ad879b7d3cfa4141bd84bb651e4ea3f187c67

oddz-contracts-master-09.07.zip(SHA256):

6a350e4ed568d1d61a659bc34170abe8818dc00ea2d7826b52638acc7b6c6c27

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Swap Path Issue	Design Logic Audit	Low	Confirmed
N2	Missing event records	Others	Suggestion	Confirmed
N3	The deflationary token docking issue	Others	Suggestion	Confirmed
N4	Token active status change issue	Design Logic Audit	Low	Fixed
N5	Redundant code	Others	Suggestion	Confirmed
N6	Unsafe External Call	Unsafe External Call Audit	Critical	Fixed
N7	Race conditions issue	Reordering Vulnerability	Critical	Fixed
N8	Excessive authority issue	Authority Control Vulnerability	Low	Confirmed

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:

PancakeSwapForUnderlyingAsset			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
swapTokensForUA	Public	Can Modify State	onlyOwner

BaseRelayRecipient			
Function Name	Visibility	Mutability	Modifiers
isTrustedForwarder	Public	-	-
msgSender	Internal	-	-

ChainlinkPriceOracle			
Function Name	Visibility	Mutability	Modifiers
setManager	Public	Can Modify State	-
removeManager	Public	Can Modify State	-
<Constructor>	Public	Can Modify State	-
getPrice	Public	-	onlyManager

ChainlinkPriceOracle			
setPairContract	Public	Can Modify State	onlyManager
setDelay	Public	Can Modify State	onlyOwner

ChainlinkIVOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mapDaysToIVPeriod	Public	Can Modify State	onlyOwner
addAllowedPeriods	Public	Can Modify State	onlyOwner
setMinVolatilityBound	Public	Can Modify State	onlyOwner
setMaxVolatilityBound	Public	Can Modify State	onlyOwner
setManager	Public	Can Modify State	-
removeManager	Public	Can Modify State	-
getIv	Public	-	onlyManager
setPairContract	Public	Can Modify State	onlyManager
setDelay	Public	Can Modify State	onlyOwner
setVolatilityPrecision	Public	Can Modify State	onlyOwner
_getVolPercentage	Private	-	-
_getNegPercentage	Private	-	-
_getPosPercentage	Private	-	-
addVolatilityMapping	Public	Can Modify State	onlyOwner

ChainlinkIVOracle			
_getIv	Private	-	-

OddzVolatility			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mapDaysToIVPeriod	Public	Can Modify State	onlyOwner allowedPeriod
addAllowedPeriods	Public	Can Modify State	onlyOwner
setMinVolatilityBound	Public	Can Modify State	onlyOwner
setMaxVolatilityBound	Public	Can Modify State	onlyOwner
getIv	Public	-	-
setPairContract	Public	Can Modify State	-
setDelay	Public	Can Modify State	onlyOwner
setVolatilityPrecision	Public	Can Modify State	onlyOwner
_getVolPercentage	Private	-	-
_getNegPercentage	Private	-	-
_getPosPercentage	Private	-	-
addVolatilityMapping	Public	Can Modify State	onlyOwner allowedPeriod
setDefaultIv	External	Can Modify State	onlyOwner
_getIv	Private	-	-

OddzAssetManager

OddzAssetManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setTimeLocker	External	Can Modify State	-
removeTimeLocker	External	Can Modify State	-
getAsset	Public	-	-
getPrecision	Public	-	-
getAssetAddressByName	Public	-	-
getPair	Public	-	-
getPrimaryFromPair	Public	-	-
getStatusOfPair	Public	-	-
getPurchaseLimit	Public	-	-
getMaxPeriod	Public	-	-
getMinPeriod	Public	-	-
validMaxDays	Private	-	-
validMinDays	Private	-	-
addAsset	External	Can Modify State	onlyOwner
activateAsset	External	Can Modify State	onlyOwner inactiveAsset
deactivateAsset	External	Can Modify State	onlyTimeLocker validAsset
addAssetPair	External	Can Modify State	onlyOwner
activateAssetPair	External	Can Modify State	onlyOwner

OddzAssetManager			
deactivateAssetPair	External	Can Modify State	onlyTimeLocker validAssetPair
updateMaxPeriod	External	Can Modify State	onlyTimeLocker validAssetPair
updateMinPeriod	External	Can Modify State	onlyTimeLocker validAssetPair
setPurchaseLimit	External	Can Modify State	onlyTimeLocker validAssetPair

OddzFeeManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setTimeLocker	External	Can Modify State	-
removeTimeLocker	External	Can Modify State	-
addTokenDiscounts	Public	Can Modify State	onlyOwner
addTxnTokens	Public	Can Modify State	onlyOwner
addSettlementTokens	Public	Can Modify State	onlyOwner
setTransactionFeePerc	External	Can Modify State	onlyTimeLocker
setSettlementFeePerc	External	Can Modify State	onlyTimeLocker
getTransactionFee	Public	-	-
getSettlementFee	Public	-	-
numDigits	Private	-	-

OddzWriteStrategy			
Function Name	Visibility	Mutability	Modifiers

OddzWriteStrategy			
<Constructor>	Public	Can Modify State	-
getPools	External	-	-
getShares	External	-	-
addLiquidity	External	Can Modify State	onlyOwner
removeLiquidity	External	Can Modify State	onlyOwner

OddzStrategyManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
createStrategy	External	Can Modify State	-
_addLiquidity	Private	Can Modify State	validPools
addLiquidity	External	Can Modify State	validStrategy
removeLiquidity	External	Can Modify State	validStrategy
changeStrategy	External	Can Modify State	validStrategy validStrategy validPools

4.3 Vulnerability Summary

[N1] [Low] Swap Path Issue

Category: Design Logic Audit

Content

In the PancakeSwapForUnderlyingAsset contract, the owner can swap fromToken to toToken through the

swapTokensForUA function. The path set is `[fromToken, toToken]`. If fromToken and toToken in PancakeSwap do not have a directly related token pair, then using this path will not be able to successfully swap.

- contracts/Integrations/Dex/PancakeSwap/PancakeSwapForUnderlyingAsset.sol#L29-L44

```
    function swapTokensForUA(
        address _fromToken,
        address _toToken,
        address _account,
        uint256 _amountIn,
        uint256 _amountOutMin,
        uint256 _deadline
    ) public override onlyOwner returns (uint256[] memory result) {
        address[] memory path = new address[](2);
        path[0] = _fromToken;
        path[1] = _toToken;
        ERC20(_fromToken).safeApprove(address(pancakeSwap), _amountIn);
        result = pancakeSwap.swapExactTokensForTokens(_amountIn, _amountOutMin, path,
address(this), _deadline);
        // converting address to address payable
        ERC20(address(uint160(_toToken))).safeTransfer(_account, result[1]);
    }
```

Solution

It is recommended to check whether fromToken and toToken have a valid token pair. If there is no token pair available, an intermediate token (such as WBNB) should be added to the swap path.

Status

Confirmed; If there is no liquidity in pancakeswap pair, the swap reverts with exception. Oddz Finance project team will not manage intermediate token paths.

[N2] [Suggestion] Missing event records

Category: Others

Content

In the ChainlinkIVOracle contract, the owner can set allowedPeriods, minVolatilityBound, maxVolatilityBound and other parameters at will, but no event recording is performed.

- contracts/Integrations/VolatilityOracle/Chainlink/ChainlinkIVOracle.sol

```
    function addAllowedPeriods(uint8 _ivAgg) public onlyOwner(msg.sender) {
        allowedPeriods[_ivAgg] = true;
    }

    function setMinVolatilityBound(uint256 _minVolatility) public
onlyOwner(msg.sender) {
        minVolatilityBound = _minVolatility;
    }

    function setMaxVolatilityBound(uint256 _maxVolatility) public
onlyOwner(msg.sender) {
        maxVolatilityBound = _maxVolatility;
    }

    function setDelay(uint256 _delay) public onlyOwner(msg.sender) {
        delayInSeconds = _delay;
    }

    function setVolatilityPrecision(uint8 _precision) public onlyOwner(msg.sender) {
        volatilityPrecision = _precision;
    }
```

Solution

It is recommended that the owner make event records when modifying sensitive parameters for follow-up self-examination and community review.

Status

Confirmed; Oddz Finance project team will relay on timelock controller events.

[N3] [Suggestion] The deflationary token docking issue

Category: Others

Content

Users can transfer the staking token into the staking contract through the deposit function. Under normal circumstances, the number of staking tokens transferred by the user is the same as the _amount parameter passed in. But if the staking token is a deflationary token, the number of tokens transferred by the user may be different from the number of tokens actually received in the contract.

- contracts/Staking/OddzStakingManager.sol, OddzTokenStaking.sol, OUsdTokenStaking

```
    function stake(IERC20 _token, uint256 _amount) external override
validToken(_token) {
        require(_amount > 0, "Staking: invalid amount");
        tokens[_token]._stakingContract.stake(msg.sender, _amount);

        emit Stake(msg.sender, address(_token), _amount);
    }

    function stake(address _staker, uint256 _amount) external override onlyOwner {
        _stake(_staker, _amount);
        _mint(_staker, _amount);

        IERC20(token).safeTransferFrom(_staker, address(this), _amount);
    }
```

Solution

If the staking token of this staking contract is a deflationary token, it is recommended to check the token balance of the contract before and after the user transfer.

Status

Confirmed; Oddz Finance project team will not have support for deflationary tokens.

[N4] [Low] Token active status change issue

Category: Design Logic Audit

Content

In the OddzStakingManager contract, the owner can set the active state of the token to true through the activateToken function, and the timelock contract can set the active state to false through the deactivateToken

function. But after the state change, the txnFeeReward, settlementFeeReward, and allotedReward parameters of each valid token did not change accordingly, so the totalTxnFee, totalSettlementFee, and totalAllotedFee parameters are not equal to 100.

- contracts/Staking/OddzStakingManager.sol

```
    function deactivateToken(IERC20 _token) external onlyTimeLocker(msg.sender)
validToken(_token) {
        tokens[_token]._active = false;
        emit TokenDeactivate(address(_token));
    }

    function activateToken(IERC20 _token) external onlyOwner(msg.sender)
inactiveToken(_token) {
        tokens[_token]._active = true;
        emit TokenActivate(address(_token));
    }
```

Solution

It is recommended that after the active state of the token changes, the txnFeeReward, settlementFeeReward, and allotedReward parameters of each valid token will also change accordingly.

Status

Fixed; This issue was be fixed in commit: cd3fa8e2d7b6fbe95a7753136338cd42e51cea4d

[N5] [Suggestion] Redundant code

Category: Others

Content

There is `_transferRewards` function in the OddzStakingManager contract, which checks whether the staker's collateral time is greater than rewardsLockupDuration, while the actual `_transferRewards` function is only called by the withdraw function and the claimRewards function.

But in both withdraw and claimRewards functions, there is a check to see if the staker's collateral time is greater than

rewardsLockupDuration. So the `_transferRewards` function does not need to check again if the staker's collateral time is greater than the rewardsLockupDuration.

- contracts/Staking/OddzStakingManager.sol

```
    function _transferRewards(
        address _staker,
        IERC20 _token,
        uint256 _date
    ) private returns (uint256 reward) {
        if (_date - tokens[_token]._stakingContract.getLastStakedAt(_staker) >=
tokens[_token]._rewardsLockupDuration) {
            reward = tokens[_token]._stakingContract.withdrawRewards(_staker);
            oddzToken.safeTransfer(_staker, reward);

            emit TransferReward(_staker, address(_token), reward);
        }
    }
```

Solution

It is recommended that there is no need to perform repeated checks in the `_transferRewards` function to save gas.

Status

Confirmed; In claimRewards function Oddz Finance project have a check for rewardsLockupDuration where as in withdraw function Oddz Finance project have a check only for lockupDuration.

[N6] [Critical] Unsafe External Call

Category: Unsafe External Call Audit

Content

_pool is entered by the user. When the user enters a malicious contract address and returns the malicious premium through the malicious contract, the tokens in the OddzLiquidityPoolManager contract can be transferred to a malicious address.

- contracts/Pool/OddzLiquidityPoolManager.sol#L300

```
function withdrawProfits(IOddzLiquidityPool _pool) external {
        uint256 premium = _pool.collectPremium(msg.sender, premiumLockupDuration);
        require(premium > 0, "LP Error: No premium allocated");

        token.safeTransfer(msg.sender, premium);
    }
```

The getSortedEligiblePools function does not check the input _liquidityParams and does not ensure that allPools is in the whitelist. When other functions depend on the data of getSortedEligiblePools, the same issues may occur.

- contracts/Pool/OddzLiquidityPoolManager.sol#L341

```
function getSortedEligiblePools(LiquidityParams memory _liquidityParams)
        public
        view
        returns (address[] memory pools, uint256[] memory poolBalance)
    {
        // if _expiration is 86401 i.e. 1 day 1 second, then max 1 day expiration
pool will not be eligible
        IOddzLiquidityPool[] memory allPools =
            poolMapper[
                keccak256(
                    abi.encode(
                        _liquidityParams._pair,
                        _liquidityParams._type,
                        _liquidityParams._model,

periodMapper[getActiveDayTimestamp(_liquidityParams._expiration) / 1 days]
                    )
                )
            ];
        uint256 count = 0;
        for (uint8 i = 0; i < allPools.length; i++) {
            if (allPools[i].availableBalance() > 0) {
                count++;
            }
        }
        poolBalance = new uint256[](count);
        pools = new address[](count);
        uint256 j = 0;
```

```
        uint256 balance = 0;
        for (uint256 i = 0; i < allPools.length; i++) {
            if (allPools[i].availableBalance() > 0) {
                pools[j] = address(allPools[i]);
                poolBalance[j] = allPools[i].availableBalance();
                balance += poolBalance[j];
                j++;
            }
        }
        (poolBalance, pools) = _sort(poolBalance, pools);
        require(balance > _liquidityParams._amount, "LP Error: Amount is too large");
    }
```

Solution

It is recommended to set up a whitelist detection mechanism to ensure that _pool must be in the whitelist address list.

Status

Fixed; This issue was be fixed in commit: 4f22eca5c9c06910ba37438e0e6db87ab2f48d4f;

[N7] [Critical] Race conditions issue

Category: Reordering Vulnerability

Content

The `enableOptionTransfer` function can be called repeatedly, When the attacker calls `enableOptionTransfer` to set a small value of _minAmount.The user does not need to enter minAmount to check when calling the optionTransfer function.Therefore, the attacker can call `enableOptionTransfer` again with a higher gas price to set a new minAmount, so that if the allowance is greater than the `minAmount + transferFee`, the user can normally execute optionTransfer calls and trade with a larger amount, and the attacker can profit.

- contracts/Option/OddzOptionManager.sol

```
function enableOptionTransfer(uint256 _optionId, uint256 _minAmount) external {
        Option storage option = options[_optionId];
        require(
            option.expiration > (block.timestamp +
```

```
assetManager.getMinPeriod(option.pair)),
            "Option not eligble for transfer"
        );
        require(option.holder == msg.sender, "Invalid Caller");
        require(option.state == State.Active, "Invalid state");
        require(_minAmount >= minimumPremium, "amount is lower than minimum
premium");

        optionTransferMap[_optionId] = _minAmount;

        emit OptionTransferEnabled(_optionId, _minAmount);
    }

function optionTransfer(uint256 _optionId) external {
    Option storage option = options[_optionId];
    require(
        option.expiration > (block.timestamp +
assetManager.getMinPeriod(option.pair)),
        "Option not eligble for transfer"
    );
    uint256 minAmount = optionTransferMap[_optionId];
    require(minAmount > 0, "Option not enabled for transfer");
    require(option.state == State.Active, "Invalid state");
    require(option.holder != msg.sender, "Self option transfer is not allowed");

    // once transfer initiated update option tranfer map
    delete optionTransferMap[_optionId];

    uint256 transferFee = _getTransactionFee(minAmount, msg.sender);
    txnFeeAggregate += transferFee;

    _validateOptionAmount(token.allowance(msg.sender, address(this)), minAmount +
transferFee);

    token.safeTransferFrom(msg.sender, option.holder, minAmount);
    token.safeTransferFrom(msg.sender, address(this), transferFee);

    address oldHolder = option.holder;
    option.holder = msg.sender;

    emit OptionTransfer(_optionId, oldHolder, msg.sender, minAmount, transferFee);
}
```

Solution

It is recommended to add a check of minAmount in the `optionTransfer` function to ensure that minAmount is consistent with expectations.

Status

Fixed; This issue was be fixed in commit: 406c84accca5ba8e2c416199e2fe6f9eb6eb31d7

[N8] [Low] Excessive authority issue

Category: Authority Control Vulnerability

Content

(1) The owner of the OddzIVOracleManager and OddzPriceOracleManager contracts can change the configuration of the contract and does not use timelock for management, there is a risk of excessive authority. The oracle affects the price of the asset. When the oracle contract is maliciously manipulated, it will cause the user's asset to be damaged.

(2) After the contracts are deployed, it is necessary to check whether TimeLocker is set correctly.

Solution

It is recommended to transfer the ownership to a timelock contract or a governance contract.

Status

Confirmed; Oddz Finance project team will change ownership to timelock controller.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0x002108030001	SlowMist Security Team	2021.07.12 - 2021.08.03	Low 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 2 critical risks, 3 low risks, 3 suggestions. And 2 low risks, 3 suggestions

were confirmed; All other findings were fixed. 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist